AFIT/GOR/ENS/98M-05

Solving Geometric Knapsack Problems using Tabu Search Heuristics

THESIS

Christopher A. Chocolaad, B.S.
1st Lieutenant, USAF

AFIT/GOR/ENS/98M-05

THESIS APPROVAL

Student: Christopher A. Chocolaad, Lieutenant, USAF Class: GOR-98M

Title: Solving Geometric Knapsack Problems using Tabu Search Heuristics

Defense Date: 3 March 1998

Committee:	Name/Title/Department	Signature
Advisor	T. Glenn Bailey, Lt Col, USAF Assistant Professor Department of Operational Sciences	T. Glenn Bailey
Reader	Richard Deckro, DBA Professor Department of Operational Sciences	Richard F. Deckro
Reader	William B. Carlton, LTC, USA Adjunct Professor Department of Systems Engineering US Military Academy	William B. Carlton

Disclaimer

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the United States Government.

AFIT/GOR/ENS/98M-05

Solving Geometric Knapsack Problems using Tabu Search Heuristics

THESIS

Presented to the Faculty of the Graduate School of Engineering of the Air Force Institute of Technology Air University In Partial Fulfillment for the Degree of

Master of Science in Operations Research

Christopher A. Chocolaad, B.S.

1st Lieutenant, USAF

Air Force Institute of Technology

Wright-Patterson AFB, Ohio

March, 1998

Sponsored in part by Air Force Studies and Analysis Agency

Acknowledgments

I would like to thank my advisor, Lt. Col. Glenn Bailey, for every thing he has taught me and for providing me with the opportunity to work on this subject matter. I am ever grateful for our conversations, his advise and support.

I am thankful for the comments, as well as insights of Lt. Col. William Carlton and Dr. Richard Deckro the readers of this thesis.

I wish to acknowledge the Air Force Studies and Analysis Agency for sponsoring and funding this topic, which with out, this research would not have been possible.

Finally, I like to thank my family,especially my mom, for encouraging and supporting me.

Table Of Contents

Page

Acknowledgments . . . iv

Table Of Contents . . . v

List of Tables . . . viii

List of Figures . . . ix

Abstract . . . x

Chapter 1. INTRODUCTION . . . 1

Chapter 2. AIR LIFT LOADING PROBLEM . . . 2

2.1 Introduction . . . 2

2.1.1 Air Lift Loading Model . . . 2

2.1.2 Tabu Search . . . 4

2.1.2.1 Memory . . . 4

2.1.2.2 Strategic Oscillation . . . 5

2.1.3 Knapsack Problems . . . 6

2.1.3.1 Single Knapsack Problem . . . 6

2.1.3.2 Multidimensional Knapsack Problem . . . 6

2.1.3.3 Geometric Knapsack Problem . . . 7

2.1.4 Packing Problems . . . 8

2.1.5 Problem Definition . . . 9

2.2 The Packing Heuristic 11

2.2.1 Simple Tabu Thresholding 11

2.2.2 STT for the Packing Problem 12

2.2.2.1 The Move Set 12

2.2.2.2 The Objective Function 14

2.2.2.3 The Candidate List Procedure 16

2.2.2.4 The Improving Phase 16

2.2.2.5 The Mixed Phase 17

2.3 The Knapsack Heuristic 17

2.3.1 Critical Event Tabu Search 17

2.3.2 Reactive Tabu Search 19

2.3.3 Computational Results 20

2.4 Geometric Knapsack Heuristic 22

2.4.1 Computational Results 23

2.5 Conclusion 23

2.6 Suggestions for Future Research 24

Appendix A. PSEUDO CODE FOR PACKING HEURISTIC 25

A.1 Improving Phase 25

A.2 Mixed Phase 25

Appendix B. PSEUDO CODE FOR KNAPSACK HEURISTIC 26

B.1 Main 26

B.2 Constructive Phase 26

B.3 Destructive Phase 27

B.4 Transfer Phase 27

Appendix C. CODE DOCUMENTATION 28

Bibliography 29

Vita 37

List of Tables

Page

Table 1. Comparison of Reactive Tabu Search with Beasley - Chu's GA 21

Table 2. Comparision of ALP Heuristics (for 10 C-17 Sorties) 23

List of Figures

Page

Figure 1. Geometric Knapsack Heuristic 22

AFIT/GOR/ENS/98M-05

Abstract

An instance of the geometric knapsack problem occurs in air lift loading where a set of cargo must be chosen to pack in a given fleet of aircraft. This paper demonstrates a new heuristic to solve this problem in a reasonable amount of time with a higher quality solution then previously reported in literature. We also report a new tabu search heuristic to solve geometric knapsack problems. We then employ our novel heuristics in a master-slave relationship, where the knapsack heuristic selects a set of cargo and the packing heuristic determines if that set is feasible. The search incorporates learning mechanisms that react to cycles and thus is robust over a large set of problem sizes. The new knapsack and packing heuristics compare favorably with the best reported efforts in the literature. Additionally, we show the JAVA language to be an effective language for implementing the heuristics. The search is then used in a real world problem of determining how much cargo can be packed with a given fleet of aircraft.

Solving Geometric Knapsack Problems using Tabu Search Heuristics

Chapter 1 - Introduction

The knapsack problem has wide application in array of industries. The problem occurs in layout, cutting stock, scheduling and budget capital contexts. It is typically described as packing as many elements of a set of items into a knapsack as possible, subject to one or more linear constraints (such as weight), in order maximize the value of its contents. The geometric knapsack problem extends this formulation by adding constraints that explicitly model the boundaries of the geometric space of the knapsack and the individual items in the knapsack such that no overlaps occur [18]. This paper introduces a new technique for solving the geometric knapsack problem used for layout or component packing. An instance of this problem occurs in air lift loading when a set of cargo must be selected for packing a given fleet of aircraft, thus establishing a strong practical interest to the existing theoretical aspects. In this context we develop a prototype heuristic to solve the air lift loading problem for the USAF Studies and Analysis Agency. The organization of this paper is as follows. Section 2.1 contains basic definitions and defines the problem. Section 2.2 describes the packing heuristic and presents results. Section 2.3 describes the knapsack heuristic and presents benchmarks against the best reported methods in the literature. Section 2.4 describes the geometric knapsack heuristic and benchmarks against USAF Studies and Analysis Windows Air Lift Loading Model.

Chapter 2 - Air Lift Loading Problem

2.1 Introduction

A difficult problem facing the United States Air Force (USAF) is accurately and efficiently planning the placement of equipment and personnel on military and Civilian Reserve Air Fleet (CRAF) aircraft. The cargo generally includes trucks, helicopters, tanks, pallets, miscellaneous equipment, hazardous material, and personnel. The aircraft moving the cargo can range from large military transports (C-5, C-17, C-141) to tactical airlifters (C-130), to CRAF airplanes (Boeing 747, Airbus 400). The matching of cargo to aircraft is referred to as a load plan, and has several competing objectives and constraints that change with different wartime scenarios. For example, NG [73] notes that a strategic mission might put priority on maximum utilization of aircraft, while a tactical mission places more emphasis on ease of off-loading cargo. Additional constraints can involve cargo height restrictions, allowable cabin load (ACL), axle weight restrictions, pounds-per-linear-foot limits, and incompatible hazardous cargo.

2.1.1 Air Lift Loading Model

Cochard and Yost [21] describe the USAF's first computer system, the Deployable Execution System (DMES) developed in 1982, for helping load planners. DMES uses a modified cutting stock heuristic suggested by Eilon and Christofides [31], and is based on Gilmore and Gomory's [37] cutting stock algorithm. DMES was rewritten and released as a standard USAF system in 1985 under the name of the Computer Aided Load Manifesting (CALM). Updates to the CALM program include migrating it to different operating systems, adding additional aircraft types, and improving the graphical user interface. No significant changes have been made to the loading heuristic itself. However, since these systems are too cumbersome for large scale airlift planning, Yost and Hare [102] developed an estimation technique for large scale planning. They compute an upper bound

with methods similar to DMES, and a lower bound with rule of thumb techniques, thus providing a worst and best case.

The USAF Studies and Analysis Agency uses the Air Lift Loading Model (ALM) to estimate airlift requirements for large scale war plans and exercise movements. ALM [95] uses one of three modified cutting stock heuristics to load vehicles (these heuristics are similar to the heuristics developed by Yost and Hare [102]). However, pallets and personnel are loaded the same way regardless of which heuristic is selected, because in actual practical settings pallets must occupy predefined positions inside the aircraft [102].

The first heuristic, *fill gap*, attempts to fill the remaining space in the cargo compartment with the next vehicle from a sorted list of vehicles. If the vehicle does not fit, the next vehicle on the list is tried. The process continued until an item is found that does. ALM then repeats this process with the next gap. The second heuristic, *top-down* ,differs from the fill-gap in that it selects the first vehicle in the list and then looks for a gap big enough to hold it, thus giving priority to the loading sequence. The third heuristic, *floor-utilization,* first sorts the vehicles by the ratio of ACL to floor space, then proceeds to use the top-down algorithm with this list.

The inherent drawbacks of these techniques are documented by Cochard and Yost [21], and Yost and Hare [102]. These heuristics only account for one objective (improving utilization of cargo), and ignore other objectives such as ease of on-off-loading and prioritized cargo. In addition, these heuristic approaches do not handle odd shaped cargo well, do not guarantee balanced loads, and have no way to add hazardous cargo constraints. Updates to ALM have been limited to migrating the program from UNIX[1] to Windows 95[2], and adding a graphical user interface. No work has been done to improve the selecting or packing heuristics themselves.

[1]UNIX is a trademark of Unix System Laboratories Inc.

[2]Windows 95 is a registered trademark of Microsoft Corporation.

2.1.2 Tabu Search

Tabu search is an intelligent problem solving approach that uses adaptive memory and responsive exploration. Its adaptive memory contrasts with most other meta-heuristics which employ either memoryless (simulated annealing and genetic algorithms) or rigid memory designs (branch and bound) [40]. The emphasis tabu search places on responsive exploration is based on the premise that a bad strategic choice will yield more information than a good random choice [44]. Tabu search has proved very effective in solving a wide range of applications and for this reason forms the foundation of this paper. We give a brief explanation of the specific tabu search characteristics we employ; however, more thorough discussions of tabu search applications and characteristics are found in [40, 43, 44].

Given a function $f(x)$ to be optimized over a set X, tabu search iteratively proceeds from one solution to another until a chosen termination criterion is satisfied. Each $x \in X$ has an associated *neighborhood* $N(x) \subset X$, and each solution $x' \in N(x)$ is reached from x by an operation called a *move*. Tabu search modifies $N(x)$ as the search progresses, effectively replacing it with a new neighborhood. Such modifications use adaptive memory with move options that can be constructive (constructive neighborhood) or destructive (destructive neighborhood). Exactly which solution to admit to the neighborhood $N^*(X)$ can be found in several ways, the most common technique being the classification of solutions within a specified horizon as "tabu" (exceptions are made if certain criteria called the *aspiration level* is met) [40].

2.1.2.1 Memory

Tabu searches can utilize two different types of memory- short and long. The most commonly used short term memory is *recency based* memory, which tracks solution attributes (as opposed to solution values), from the immediate history of the search. Attributes that appear in recent solutions

become tabu active, while solutions containing some combination of tabu active attributes become tabu themselves. This prevents solutions recently visited from belonging to $N^*(X)$ while at the same time admitting new solutions with the desired characteristics [44].

Short term memory alone has the ability to produce high quality solutions; however, the literature shows long term memory can substantially improve the search, even for short solution runs [42]. The fundamental technique for implementing the long term approach is *frequency based* memory, which tracks the relative span any particular attribute has belonged to solutions, then penalizes or rewards potential solutions. Two important concepts of long term memory are intensification and diversification strategies. Intensification strategies encourage move choices in the regions that have historically produced good solutions, while diversification strategies drive the search into unexplored areas of X.

2.1.2.2 Strategic Oscillation

One method of balancing intensification and diversification strategies is *strategic oscillation* [43]. Strategic oscillation directs the search towards a critical condition that would otherwise stop the search. However, strategic oscillation forces the search past the critical condition to a specified level, then allows the search to return to the critical condition. An example of using strategic oscillation is when the critical condition is defined as feasibility; once the boundary of feasibility is reached the search will continue a select number of steps into the infeasible region before returning to the feasible region (or vice-versa). The criteria for choosing the next move differs based on whether the current solution is feasible.

2.1.3 Knapsack Problems

2.1.3.1 Single Knapsack Problem

The *single knapsack problem* or the *zero-one knapsack problem* (KP) models the selection of n items with weight w and value p to be packed in a container of capacity b such that we:

$$\text{Maximize} \sum_{j=1}^{n} p_j x_j$$

subject to

$$\sum_{j=1}^{n} w_j x_j \leq b \tag{1}$$
$$x_j \in \{0,1\}.$$

Martello and Toth [67] show KP to be NP-hard and provide a detailed discussion of this class of problems as well as algorithms and heuristics to solve them.

2.1.3.2 Multidimensional Knapsack Problem

The *multidimensional knapsack problem* (MDKP), is a NP-hard problem with the same formulation as the KP except (1) is substituted with

$$\sum_{j=1}^{n} w_{jk} x_j \leq b_k, \qquad k \in Q\,\{1, ..., q\} \tag{2}$$

where q is the number of constraints. This can also be referred to as the *loading problem,* where several different lengths of material are packed into vessels of fixed capacities. While the loading problem can have many dimensions (e.g. length, weight, volume) the literature often assumes the capacity requirements are additive [26, 31]. Therefore when packing a container under a volume constraint, the container must be free to conform to the shape of the packed items, or conversely the items must be fluid to conform to the shape of the container. Chu and Beasley [20] review in detail both algorithms and heuristics to solve the MDKP. They note that effective optimal solution algorithms have only been demonstrated on problems where q is relatively small. For problems where n and q are both large, existing exact and heuristic methods are of limited effectiveness.

Two new heuristics, a critical event tabu search by Glover and Kochenberger [42] and a genetic algorithm by Chu and Beasley [20], show promise in solving problems of larger size. While neither directly compare the two heuristics, both demonstrate great improvement over previous methods in CPU time and solution quality.

2.1.3.3 Geometric Knapsack Problem

The KP and MDKP do not address the *geometry* of either the container or individual the items. In other words, the *shape* of an item, and how that shape affects its ability to fit in the container is not captured in MDKP. The *geometric knapsack problem* (GKP) extends the MDKP by explicitly modeling the shape of each item and the container – in effect, removing the additivity relaxation. For example, in one version of the GKP the position of the items is fixed; then, a optimal container enclosing some subset of those items is selected [5].

In the present problem we consider the space and dimensions of the container as fixed with no items overlapping. The formulation repeats KP with two additional constraints. Following Cagan [18] let S_{total} be the space (location and volume) bounding the container volume in $\Re^3$. Also, let $S(x_j)$ and $S(x_k)$ be the space of the j and k cargo items, respectively, in $\Re^3$ such that

$$S(x_j) \cap S(x_k) = \emptyset \qquad \forall j \neq k \tag{3}$$

$$S(x_j) \subseteq S_{total} \qquad \forall x_j. \tag{4}$$

Equation (3) states that one item can not occupy the same space as the other while (4) ensures the items must be inside the container.

The heuristic techniques in the literature for the KP and MDKP are not effective for the GKP because of the added geometric complexity. Cagan's shape annealing heuristic combines the formalism of shape grammar that dictates permissible item orientation with simulated annealing. However, we need a heuristic that allows a more robust set of item orientation; thus, our approach to

GKP problem is to decompose it into a KP and a packing problem. The KP heuristic selects the set of items to potentially pack while the packing heuristic optimizes the placement of the selected items inside the knapsack. The solution found from the packing problem provides the updated constraint vector to the KP.

2.1.4 Packing Problems

In surveys of packing problems conducted by Coffman *et al.* [30], Dyckhoff [29], and Dowsland and Dowsland [26], the majority of literature deals with lower dimensional packing problems with regular shaped objects. Dowsland and Dowsland point out that the rectangular packing problem is NP-complete; thus, non-rectangular problems are often not pursued due to the increasing complexity. They also note that for three-dimensional problems, most approaches employ ad-hoc rules based on common sense; resulting in, no single approach being seen as superior. Furthermore, practical experience shows that while these methods for three dimensional problems will out perform manual methods on average, they are computationally expensive. Finally, Dowsland and Dowsland note that a concerted manual effort will beat these algorithms in terms of packing density [26].

A recent exception to these heuristics for the three dimensional packing is the area of mechanical design. Szykman and Cagan [85] extend the simulated annealing technology for two dimensional VLSI layout by developing a simulated annealing based approach to packing three dimensional objects into a container. They also employ their method to solve the three dimensional component layout problem with the objective of achieving high packing density subject to fitting components into a container that satisfies separation constraints. While similar to our need of packing an aircraft at a high density while maintaining the separation constraints on the cargo, our approach differs in that we maintain a balanced load on each aircraft and employ a tabu search meta-heuristic.

2.1.5 Problem Definition

Given a fleet of aircraft, how much cargo can be moved? Answering this question requires two decisions: which cargo to place in each aircraft and the cargo's placement inside. Selecting cargo recalls the knapsack problem, where each piece of cargo has weight, volume, and value, while the aircraft have a finite volume and weight limitation. Given m aircraft and a set of n cargo items with a value p, the problem formulation is:

$$\text{Maximize} \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \tag{5}$$

Subject To

$$\sum_{j=1}^{n} W_{c_j} x_{ij} \leq W_{payload_i} \qquad i \in M = \{1, ..., m\} \tag{6}$$

$$\sum_{j=1}^{n} V_{c_j} x_{ij} \leq V_{payload_i} \qquad i \in M = \{1, ..., m\} \tag{7}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1 \qquad j \in N = \{1, ..., n\} \tag{8}$$

$$x_{ij} = \text{binary} \qquad i \in M\,, j \in N \tag{9}$$

where

$$x_{ij} = \begin{cases} 1 \text{ if cargo item } j \text{ is assigned to aircraft } i; \\ 0 \text{ otherwise;} \end{cases} \tag{10}$$

Formulation (5-10) without (7) is the multiple knapsack problem (MKP), shown by Martello and Toth [67] to be in the NP-hard class of problems. Since the addition of constraint (7) makes the problem multidimensional, we refer to (5-10) as the multidimensional multiple knapsack problem (MMKP).

Arranging the set of cargo items selected for each aircraft imposes additional constraints on MMKP, since the available space and location of where cargo may be placed is fixed and cargo cannot overlap. Following Cagan [18], let S_{total_i} be the space (location and volume) bounding the payload volume in $\Re^3$ of aircraft i. In addition, let $S(x_{ij})$ and $S(x_{ik})$ be the space of the j and k

cargo items, respectively, in aircraft i in $\Re^3$ such that

$$S(x_{ij}) \cap S(x_{ik}) = \emptyset \qquad \forall j \neq k \tag{11}$$

$$S(x_{ij}) \subseteq S_{total_i} \qquad \forall x_{ij}. \tag{12}$$

Equation (11) states that no cargo item can occupy the same space as another, while (12) restricts individual cargo items to fitting within the space of the corresponding aircraft.

We call the new formulation (5-12) the geometric multidimensional multiple knapsack problem (GMMKP). We now extend the GMMKP formulation to the Air Loading Problem (ALP). First, payload restrictions vary by location due to different floor strengths. Therefore let t be a section of aircraft i that can sustain a maximum floor load of P_{t_i}, $P(x_{ij})$ denote the loading of cargo item j, and $S(t_i)$ the space section t_i occupies inside aircraft i such that

$$S(t_i) \cap S(x_{ij}) = \emptyset \qquad \forall P(x_{ij}) > P_{t_i}. \tag{13}$$

Second, some cargo items must have separation constraints; e.g., two trucks cannot sit next to each other. Let D_{jk} be the distance required between cargo items j and k, and define the function $L\left[S(x_{ij}), S(x_{ik})\right]$ as the distance between cargo item j and k on aircraft i such that

$$L\left[S(x_{ij}), S(x_{ik})\right] \geq D_{jk} \quad \forall j \neq k. \tag{14}$$

Third, packing arrangements must not cause the aircraft to destabilize by shifting the aircraft's center of gravity (c.g.) outside its design limits. Letting L_{cg_i} be the location of aircraft i's c.g. when packed, and $L_{design_{max\,i}}$ and $L_{design_{min\,i}}$ be the location of the aircraft i's maximum and minimum design c.g., respectively,

$$\begin{aligned} L_{cg_i} &< L_{design_{max_i}} \\ L_{cg_i} &> L_{design_{min_i}}. \end{aligned} \tag{15}$$

We call the GMMKP with constraints (13-15) the ALP.

2.2 The Packing Heuristic

Theodoracatos and Grimsley [90] note that since the general packing problem belongs to the NP-complete class of problems, and typically contains a large number of sub-optimal solutions, a meta-heuristic is needed. Szykman and Cagan [85] use a simulated annealing approach to solve a similar problem of three-dimensional component packing, while Theodorcatos and Grimsley use simulated annealing to pack arbitrarily shaped polygons. However, Dowsland's [27] experiment with Glover's [41] simple tabu thresholding on the rectangular packing problem shows promising results, thus motivating our use of simple tabu thresholding to solve the packing portion of the ALP.

2.2.1 Simple Tabu Thresholding

Simple tabu thresholding (STT) is a local search method that avoids becoming trapped at local optimum by allowing non-improving moves. A successful implementation requires a well defined solution space, neighborhood structure and cost function. Glover [41] presents a detailed description of this method; only a brief overview is given here. STT combines strategic oscillation with a candidate list strategy. Strategic oscillation refers to the technique of orienting moves in relation to a critical condition, and the candidate list strategy refers to the method used to pick the moves. The STT method differs from other tabu search methods in that it has a greatly reduced reliance on memory. Instead, it controls randomization using a candidate list strategy to fulfill functions otherwise provided by memory; assigns probabilities to reflect evaluations of attractiveness by weighting over near best intervals; and, judiciously selects the subset of moves from which intervals are drawn [44].

STT consists of two alternating phases, an improving phase and a mixed phase. Both phases partition the neighborhood moves into subsets, and only one subset is considered at each iteration. The improving phase only accepts moves that improve the objective function (see A.1), while the

mixed phase (see A.2) accepts all moves. During the improving phase, a block random order scan (BROS) chooses the subsets to search. BROS allocates each subset a position in a cyclic list, with a total of M subsets. The improving phase searches the list sequentially, starting over again once the cycle has been completed. BROS groups the subsets into k blocks; when the improving phase encounters each block, the BROS shuffles the elements of that block. As long as k does not divide M, the BROS permits the resequenced elements to migrate. This effectively avoids cycling by emulating a tabu list of approximately M [41]. The improving phase terminates when reaching a local optimum, thus initiating the mixed phase.

The mixed phase begins by selecting a random tabu timing parameter t between the specified limits of t_{min} and t_{max}, and conducts a full random order scan (FROS) of M. FROS shuffles all of the subsets M, ignoring the block groupings of the improving phase. The mixed phase searches the list sequentially; if the mixed phase reaches the end of the list (this will only occur if t is greater than M), a BROS selects the remaining subsets to be searched. This phase continues for t iterations, or until an aspiration criteria is satisfied.

2.2.2 STT for the Packing Problem

In this section, we describe the STT packing heuristic. The packing heuristic checks the feasibility of the MMKP. The knapsack heuristic then uses the solution of the packing heuristic as the updated right hand side vector.

2.2.2.1 The Move Set

We base our move sets on Dowsland [27], where the neighborhood moves are apportioned by assigning one subset to each cargo item in the layout; thus subset j contains all possible moves for cargo item j. While the basic moves are borrowed from Szykman and Cagan [85], our STT differs from their simulated annealing approach in that we evaluate each move before making it, and only

accept improving moves during the improving phase. We employ three types of moves in each subset to perturb the layout- - *translate*, *rotate* and *swap* moves.

Translate. Each translate move has a distance D associated with it, where D ranges from a minimum to a maximum value (multiple translate distances allows the algorithm to evaluate steps of varying size). Theodorcatos and Grimsley [90] observe that the objective function for the two-dimensional packing problem is based upon a polygonal area consisting of a bounding box and penalties for polygonal overlap. Their experience with the their simulated annealing heuristic suggest the size of the neighborhood set should be based upon the sum of the polygonal areas of the cargo items. They provide the following relation to set the initial maximum distance for the two-dimensional problem:

$$D_{\max} = \sqrt{\frac{\sum_{j=1}^{n_i} Area_{c_j}}{\pi}} \tag{16}$$

When packing aircraft, cargo is not stacked on top of each other, so we limit translation of cargo items to width and length directions. When evaluating a translate move, a cargo item is placed at distance $D \bullet \mathbf{V}$, where $\mathbf{V}$ is defined as a unit vector.

Rotate. We limit the rotations to the vertical axis with three defined moves of 90, 180, and 270, degrees. In general, cargo can rotate a full 360 degrees; however, for those cargo items that must rest inside the aircraft in a certain orientation the rotation is limited accordingly.

Swap. Swap moves switch an item's centroid location. We employ one swap move in the improving phase and multiple swaps in the mixed phase.

The cargo items all come from a standard database enabling us to model each cargo item as a separate object using the object-oriented language JAVA[3]. By developing a separate class for each general shape of cargo item, we enable each type to have a distinctive move set based on these three categories.

[3] Java is a trademark of Sun Microsystems, Inc.

2.2.2.2 *The Objective Function*

Following Szykman and Cagan [85], our STT uses a multiple objective function F of the weighted sum form

$$F = W_{o1}f_1 + W_{o2}f_2 + ... + W_{op}f_p \tag{17}$$

where f_l is the value of the lth objective and W_{ol} is the weight for the lth term. Maximizing packing density constitutes the first term of the objective function

$$f_1 = \frac{S_{bb}}{\sum_{j=1}^{n_i} S_{c_j}}$$

where S_{bb} is the area of the bounding box of the packed cargo, n_i is the number of cargo items in aircraft i, and S_{c_j} is area of the jth item. By minimizing the area the cargo occupies more cargo items are packed into each aircraft, thus enabling higher values of (5). At each move cargo items are allowed to overlap each other, permitting a more thorough search of the state space. To satisfy (11) we employ a penalty function for overlap as our second term

$$f_2 = \sum_{j=1}^{n_i-1} \left(\sum_{k=j+1}^{n_i} O_{jk}^2 \right) \tag{18}$$

where O_{jk} is the overlap between the jth and kth item. For simple shapes such as rectangular blocks and cylinders, rapid geometric interference testing is possible by taking advantage of the Manhattan geometry (where all objects are oriented perpendicular to each other) [87]. Generic shapes, however, require more robust methods of computing geometric intersection.

For the two-dimensional case we model the cargo items as simple polygon objects (no overlapping edges allowed and not restricted to being convex). When each cargo item is instantiated, we decompose or *triangulate* the cargo's shape into $v-2$ triangles (where v is the number of vertices of the polygon) and store the resulting triangles as arrays of triangle objects. We triangulate the cargo items by coding a JAVA version of Narkhede and Manoch [70] triangulation code, which is an $\mathbf{O}(v \log v)$ incremental randomized algorithm that in practice exhibits near linear time. We then employ

the methods described in Theodoracatos and Grimsley [90], Sedgewick [82], Foley *et al.* [33], and Preparata [76] to compute the areas of overlap during the execution of the packing algorithm.

The third component of the objective function penalizes violations of (12) i.e., (items that protrude from the aircraft) with the function

$$f_3 = \sum_{j=1}^{n_i} P_j^2$$

where n_i is the number of cargo items in the aircraft i, and P_j is the protrusion of cargo item j from the aircraft given by

$$P_j = P_{xj} + P_{yj} + P_{zj}$$

where P_{xj}, P_{yj}, and P_{zj} are the lengths of protrusion of the jth cargo item in the X, Y, Z coordinate directions, respectively.

Center of gravity (c.g.) calculations are made for the longitudinal axis only because c.g. changes along the vertical or lateral axis are small and flight controls can compensate for any effect on the stability of the aircraft. However, a longitudinal change in c.g. can cause aircraft instability. For a detailed explanation of aircraft stability see Roskam [78]. We penalize violations of (15) with a function based on the work of Amiouny *et al.* [3]

$$f_4 = d_{xj}^2$$

where d_{xj} is the distance cargo item j would have to move to put aircraft i's c.g. inside the parameters of $L_{design_{Max}}$ or $L_{design_{\min}}$. We calculate d_{xj} using conservation of momentum under the assumption that the aircraft and cargo moments are in equilibrium. Specifically

$$d_{xj} = \frac{W_{Total_i} L_{c.g._{Design}} - \sum_{k=1,k\neq j}^{n} W_{ck} L_{c.g._{cargo\,k}}}{W_{cj}} \tag{19}$$

where W_{Total} is total weight of aircraft i with cargo items j, and $L_{c.g._{cargo_j}}$ is the location of cargo item j's c.g. We assume aircraft g-load is constant and that the items are homogenous; therefore, the force from an individual item is a point load at the centroid of the item. Other loading heuristics in

the literature that consider balance are [3,15,98]. Amiouny *et al.* show the one dimensional balance problem is strongly NP-complete and propose a heuristic based on moments. Wodziak and Fadal [98] use a genetic algorithm to pack a balance load on a truck. Brosh [15] allocates cargo aboard a civilian airliner using a sequence of linear programming problems whose solutions converge to the optimum.

2.2.2.3 The Candidate List Procedure

Integers between 0 and $n_i - 1$, representing the move set of each cargo item assigned to aircraft i, populate the candidate list. The improving phase uses BROS to select moves, where the block size for aircraft i is the $Minimum(n_i, 5)$ when $n_i < 100$; otherwise, the block size is $\frac{n_i}{20}$. At the beginning of the mixed phase, STT makes a FROS of the candidate list; if t is greater then n_i, the process reverts to a BROS after n_i moves. Furthermore, our JAVA implementation represents the candidate list procedure as an object. This allows the parameters of the candidate list procedure to change at run time using the above logic, thus enabling concurrent packing heuristics to run.

2.2.2.4 The Improving Phase

The improving phase evaluates all potential moves in each cargo items move set, and selects the overall best move based on the objective function value. We decrease $D_{\max}$ at each iteration of the improving phase based on the observation that as cargo items are packed more tightly, the distances of improving moves decreases. If no improving moves are found in n_i iterations, STT exits the improving phase. If the current objective function value is the best found, STT keeps the location and position of the cargo items.

2.2.2.5 The Mixed Phase

At the start of the mixed phase, $D_{\max}$ is set to the original value found using (16). STT selects a random move for each move subset visited, and exits the mixed phase after t iterations, or if the move results in the best solution found so far.

2.3 The Knapsack Heuristic

We solve a MDKP problem to obtain an upper bound on the ALP. The ALP has thousands of items to be packed; however, there are only slightly more than 600 different types of items to choose from, thus effectively setting the maximum number of columns that will need to be updated to 600. The volume of the items is not additive (due to shape) so we substitute total length for volume in the relaxed problem. Additionally, we add a final constraint that limits the number of pallets to be packed on the aircraft. The relaxed problem will then be a MDKP with a q of three and an effective n of 600. The literature shows tabu search and genetic algorithms to be the most promising techniques to use to solve MDKPs [9,20,42,50,74]. In the literature Chu and Beasley's [20] genetic heuristic, and Glover and Kochenberger [42] tabu search show the best results in terms of solution quality and time for large MDKPs. Battiti and Tecchiolli [8] present a reactive scheme that increases the performance of strict tabu search, thus motivating us to investigate a new heuristic that combines Glover and Kochenberger's critical event tabu search with Battiti and Tecchiolli's reactive tabu scheme.

2.3.1 Critical Event Tabu Search

Glover and Kochenberger's [42] critical event tabu search uses strategic oscillation to alternate between constructive and destructive phases (see B.1). The constructive phase adds items to the knapsack while the destructive phase removes them. The search oscillates around the feasibility boundary for *span* moves; starting at one span it increases to a limiting value, then returns to

one. The pattern repeats for a set number of total *outer oscillations*. A critical event is the last solution obtained before the search entering the infeasible region in the constructive phase, or the first feasible solution after leaving infeasible space in the destructive phase. The parameters $p1$ and $p2$ in the transfer phase (see B.4) control the amount of diversification of the search. Large values of $p1$ and $p2$ provide greater diversity by forcing the heuristic to search further away from the feasibility boundary; conversely, small values encourage the heuristic to focus the search around the last critical event. Recency and frequency information influence which items to add or drop in the constructive and destructive phases. Recency information is stored in a first-in first-out queue of length $tabuTenure$. When adding a solution to the queue the variable $TABU\_R_j$ increases by one for each item j that composes the critical solution. Similarly $TABU\_R_j$ decreases by one once the solution leaves the queue. Frequency information is tracked in a similar manner; parameter $TABU\_F_j$ increases by one for each item j that is a member of a critical solution. Parameter k manages the number of tabu-influenced add or drop moves made immediately after a critical event by starting at one and increasing by one after $2 \times tabuTenure$ moves until reaching the constant $KMAX$. At this point k resets to one and repeats the process.

The variable $RATIO_j$ is the ratio of $profit$ to $surrogate_j$ where $surrogate_j$ is the surrogate constraint of item j. The heuristic utilizes three different surrogates depending on the feasibility status of the current solution. The constructive phase (see B.2) chooses an item to add to the container by selecting either the item that maximizes (20) when $count\_var > k$, or maximizes (21) when $count\_var \leq k$.

$$(RATIO_j, \qquad j \in x = 0) \tag{20}$$

$$(RATIO_j - PEN\_R \times TABU\_R_j - PEN\_F \times TABU\_F_j, \qquad j \in x = 0). \tag{21}$$

The destructive phase (see B.3) chooses an item to drop by minimizing (20, 21) using the same criteria. We define PEN_R and PEN_F as

$$PEN\_R = Maximum(RATIO_{j_{initialization}})$$

$$PEN\_F = \frac{PEN\_R}{100000 \times iterationCount}.$$

Aspiration criteria generates two additional trial solutions at a critical event. In the constructive phase, the search arrives at a point where the next move brings the heuristic to the infeasible region. When such a move is imminent, candidate items are searched in order of decreasing profit for the first one that can be added to the container while maintaining feasibility. A second solution is then generated by retaining the regularly selected move that brought the heuristic to the infeasible region, then searching for an item to drop in order of increasing profit. The trial solutions do not replace the standard move choice; they just provide a solution for use if it improves the best one currently known.

2.3.2 Reactive Tabu Search

Glover and Kochenberger start with initial values $p1 = 3, p2 = 7,\ t = 7$ and run the heuristic for a fixed amount of outer oscillations. They then modify the parameters and restart the search, recording the best solution obtained. Battiti and Tecchiolli [8] propose a fully automated reactive mechanism for on-line determination of free parameters, thus allowing the heuristic to cover a wide variety of problems while avoiding human trial and error adjustment [7]. They show in [9] the reactive search is robust and efficient for multidimensional knapsack problems of both large and small sizes. We adapt Battiti and Tecchioli's technique to Glover and Kochenberger's search. The heuristic stores the critical events visited during the search and corresponding iteration numbers in memory, so that after the last critical event one can check for repetition of critical solutions and calculate the intervals between them. When the repetition of a critical event is greater than REP,

the $tabuTenure$ is geometrically increased by ten percent. The number of iterations executed after the last change in $tabuTenure$, $stepsSinceLastChange$, is then compared to the moving average of the detected cycle length, $movingAverage$; if the $stepsSinceLastChange$ is greater than $movingAverage$ the $tabuTenure$ is decreased by ten percent. The variable $chaotic$ tracks the number of often repeated critical events; if $chaotic$ is greater than the constant $CHAOS$ an escape sequence initiates.

2.3.3 Computational Results

We benchmark our results with problems obtained from [11] to demonstrate that our heuristic is competitive in terms of quality and speed, and to show that a program written in JAVA does not significantly affect the speed of the implemented heuristic. Our reported data was run on a Digital DEC ALPHA with 64 megabytes of memory and a processor speed of 125mHz using SUN Microsystems Just-In-Time compiler 1.1.4. (We also note that the code also ran on x86 and Sun Sparc platforms with no debugging or additional coding). Table 1 compares our implementation to Chu and Beasley's genetic algorithm.

Table 1. Comparison of Reactive Tabu Search with Beasley - Chu's GA

Problem			Chu Beasley (C Code)				Reactive(JAVA)			
q	n	α	Average % Gap	A.B.S.T	A.E.T	NOPT	Average % Gap	A.B.S.T	A.E.T	NOPT
5	100	0.25	0.99	9.6	345.9	10	1.20	17.2	45.8	
		0.50	0.45	23.5	347.3	10	0.56	23.7	43.1	
		0.75	0.32	26.9	361.7	10	0.43	13.6	41.3	
5	250	0.25	0.23	50.7	682.0	8	0.38	40.0	115.7	
		0.50	0.12	276.7	709.4	5	0.23	45.5	107.8	
		0.75	0.08	195.9	763.3	5	0.13	34.6	102.1	
5	500	0.25	0.09	264.6	1271.9		0.20	90.9	239.7	
		0.5	0.04	291.3	1345.9		0.11	127.7	224.9	
		0.75	0.03	386.2	1412.6		0.06	91.7	210.6	
10	100	0.25	1.56	97.5	384.1		2.16	31.0	57.8	
		0.5	0.79	97.3	418.9		1.15	17.2	54.1	
		0.75	0.48	16.8	462.6		0.69	30.0	51.4	
10	250	0.25	0.51	359.0	870.9		0.93	76.2	153.7	
		0.50	0.25	342.2	931.5		0.53	84.5	136.6	
		0.75	0.15	129.1	1011.2		0.29	31.7	128.6	
10	500	0.25	0.24	702.5	1504.9		0.46	156.1	315.3	
		0.50	0.11	562.2	1728.8		0.25	132.9	282.7	
		0.75	0.07	937.6	1931.7		0.15	131.6	262.8	
30	100	0.25	2.91	177.4	604.5		3.72	42.7	105.7	
		0.50	1.34	118.0	782.1		1.97	39.7	95.7	
		0.75	0.83	90.1	904.2		1.19	28.4	93.6	
30	250	0.25	1.19	582.9	1499.5		2.06	114.958	262.1	
		0.50	0.53	901.5	1980.0		1.04	69.064	233.1	
		0.75	0.31	1059.3	2441.4		0.57	102.298	223.5	
30	500	0.25	0.61	1127.2	2437.7		1.14	294.7	552.4	
		0.50	0.26	1121.6	3198.9		0.54	130.9	488.1	
		0.75	0.17	1903.3	3888.2		0.32	196.2	460.9	

A.B.S.T = average best-solution time(CPU seconds)
A.E.T = average execution time (CPU seconds)
NOPT = number of instances (out of ten) the heuristic finds the optimal solution
Chu-Beasley's GA run on a Silicon Graphics Indigo workstation (R4000,100MHz,48Mb main memory)
Reactive Tabu run on a Digital DEC Alpha
(125MHz, 64MB main memory using a JIT 1.1.4 Java Compiler)

Table 2 shows the result of our upper bound knapsack heuristic on three reference ALP's verses the USAF's Windows ALM model. Each reference set has up to 610 different types of items

and up to 10,000 total items. The knapsack heuristic ran in a loop until either all available cargo was packed, or until it ran out of available aircraft.

2.4 Geometric Knapsack Heuristic

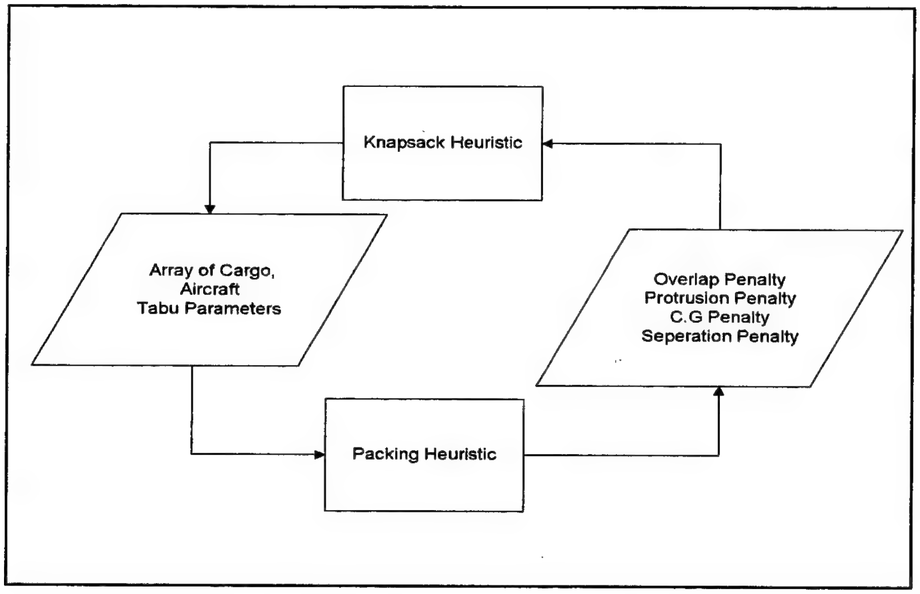

Figure 1. Geometric Knapsack Heuristic

The Geometric Knapsack Heuristic (GKH) combines the knapsack and packing heuristic together in a master slave relationship (see figure 1). The knapsack heuristic selects potential cargo to pack and the packing heuristic finds the optimal packing pattern for the selected cargo. The only change to the knapsack heuristic is in how it updates the resource constraints. The constraints that are additive (for the ALP these would be weight and maximum number of pallets) are calculated in the same manner as before but for non additive constraints (for the ALP these would be non-protrusion (12), non-overlap (11), non c.g. violation (15), and non-separation violation (14)) the packing heuristic is called. The right hand side for the non-additive constraints are initialized to zero. If the best solution (recall we have not establish optimality) to the packing heuristic violates any one of the non additive constraints the penalty from packing heuristic is subtracted from the re-

source vector of the knapsack heuristic. When the knapsack heuristic enters the destructive phase, it will drop the cargo with violations because of (20, 21).

2.4.1 Computational Results

Table 2 shows the three reference ALP problems with the GKH as compared to the results of Windows ALM (currently being used by USAF Studies and Analysis Agency). The MDKP is an upper bound on the ALP since it only considers aggregate area and weight. We note that all of the solutions obtained with our GKH are feasible, and on average less than half the equipment recommended by ALM should be loaded. These results suggest that ALM may be overestimating the amount of cargo that can be carried in a C-17, due to neglecting center of gravity limits. Further analysis of the ALM loads need to be conducted to prove this hypothesis.

Table 2. Comparision of ALP Heuristics (for 10 C-17 Sorties)

Problem Size			Equipment Taken
9398	ALM		92
	MDKP	(upper bound)	452
	GKH		35
2711	ALM		75
	MDKP	(upper bound)	286
	GKH		62
9398	ALM		92
	MDKP	(upper bound)	1451
	GKH		16

2.5 Conclusion

We introduce a novel approach to solving geometric knapsack problems, using new tabu heuristics for both the packing and multidimensional knapsack problem that compare favorably with results reported in the literature. Our approach is effective for solving the real world problem of determining which set of cargo to load aboard a given fleet of C-17 aircraft. Finally, we confirm the use of JAVA as a programming tool for heuristic applications.

2.6 Suggestions for Future Research

Develop a knapsack heuristic that will handle the packing of multiple knapsacks. Currently, the knapsack heuristic only finds the best set of items for a single knapsack. The possibility exists that a load master will have the opportunity to pack multiple aircraft at once. The multiple knapsack heuristic would pick the best set of items for all the knapsacks.

The data for ALM is in flat files. Migrating this data to a relational data base would allow easier manipulation of the data. Potentially load masters could change the value of an item in real time, enabling last minute changes to deployments to be analyzed.

Parallel implementation of the heuristics, similar to [74], would provide a way to potentially reduce the solution times of the heuristics. This would be particularly useful if the parallel implementation uses existing processors and tied them together through world wide web.

Incorporating a fast collision detection algorithm for three dimensional non-convex objects would be a valuable improvement: The up coming release of Java 1.2 with the new 3-D API may provide an easy method for doing this.

Explore using the packing heuristic on engineering design problems, like [87] does with there simulated annealing packing heuristic.

Implement the packing heuristic on the world wide web for USAF load masters to evaluate and potentially use. This would provide a cheap and innovative way of validating the heuristic with respect to the air loading problem.

APPENDIX A - Pseudo Code For Packing Heuristic

A.1 Improving Phase

Procedure 1 Improving Phase

```
while Not at local Optimum do
  Apply Candidate List Strategy by a Block Random Order Scan
  if move is improving then
    accept move
  end if
end while
```

A.2 Mixed Phase

Procedure 2 Mixed Phase

```
Select a tabu timing parameter t
for i ⇐ 0, i < t do
  Apply Candidate List Stategy by a Full Random Order Scan
  automatically accept move
end for
```

APPENDIX B - Pseudo Code For Knapsack Heuristic

B.1 Main

Procedure 3 Main

Require: $Intializeallx \Leftarrow 0$
Require: $feasible \Leftarrow true$
  Choose values for p1 and p2
  while $outeroscilliations \leq MAXOSCILLIATION$ **do**
    $constructivePhase()$
    $transferPhase()$
    $destructivePhase()$
    $transferPhase()$
    $outerOscilliations \Leftarrow outerOscilliations + 1$
  end while

B.2 Constructive Phase

Procedure 4 Constructive Phase

  $countSpan \Leftarrow 0$
  while $feasible = true$ **do**
    if no component of x_j of x can be increased from 0 to 1 except by violating feasiblity **then**
      if $cx > cx^*$ **then**
        $x^* \Leftarrow x$
      end if
      $feasible \Leftarrow false$
    else
      choose an x_j to increase from 0 to 1 such that the move maintains feasiblity
    end if
  end while
  while $feasible = false$ **do**
    $countSpan \Leftarrow countSpan + 1$
    if $countSpan > span$ or all $x_j = 1$ **then**
      $return$
    else
      choose an x_j to increase from 0 to 1
    end if
  end while

B.3 Destructive Phase

Procedure 5 Destructive Phase

$countSpan \Leftarrow 0$
while $(feasible = false)$ **do**
  select an x_j to change from 1 to 0
  if solution is feasible **then**
    if $cx > cx^*$ **then**
      $x^* \Leftarrow x$
    end if
    $feasible \Leftarrow true$
  end if
end while
while $(feasible = true)$ **do**
  $countSpan \Leftarrow countSpan + 1$
  if $countSpan > span$ or all $x_j = 1$ **then**
    $return$
  else
    choose an x_j to decrease from 1 to 0
  end if
end while

B.4 Transfer Phase

Procedure 6 Transfer Phase

if $increasingSpan = true$ **then**
  if $(span \leq p1)$ and $(p2 \times span\ outerOscillilations$ **then**
    $span \Leftarrow span + 1$
  else if $(increasingSpan = true)$ and $(span > p1)$ and $(p2\ outerOscillilations)$ **then**
    $span \Leftarrow span + 1$
    if $span > p2$ **then**
      $increasingSpan \Leftarrow false$
      $p2 \Leftarrow span - 1$
    end if
  end if
else
  if $(span > p1)$ and $(p2\ outerOscillilations)$ **then**
    $span \Leftarrow span - 1$
  else if $(span \leq p1)$ and $(p2 \times span\ outerOscilliations)$ **then**
    $span \Leftarrow span - 1$
    if $span < 1$ **then**
      $increasingSpan \Leftarrow true$
      $span \Leftarrow span + 1$
    end if
  end if
end if

APPENDIX C - Code Documentation

Class Hierarchy

- class java.lang.Object
  - class AFIT.Alm.Packing.CandidateListStrategy
  - interface AFIT.Alm.Packing.Cargo
  - class AFIT.Alm.Packing.Cargo2d (implements AFIT.Alm.Packing.Cargo)
    - class AFIT.Alm.Packing.Helicopter
    - class AFIT.Alm.Packing.Vehicle
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Canvas
      - class AFIT.Alm.Packing.PackCanvas
      - class AFIT.Alm.Packing.PackingCanvas
  - class AFIT.Alm.Packing.Container
    - class AFIT.Alm.Packing.BalancedContainer
      - class AFIT.Alm.Packing.Aircraft
        - class AFIT.Alm.Packing.SectionedAircraft
          - class AFIT.Alm.Packing.C17
  - class AFIT.Alm.Knapsack.EquipmentAlm (implements java.io.Serializable)
  - class AFIT.Alm.Knapsack.Reader.EquipmentReader
  - class AFIT.Alm.Geometry.Geometry2d
  - class AFIT.Alm.Knapsack.GroupAlm (implements java.io.Serializable)
  - class AFIT.Alm.Knapsack.ID (implements java.io.Serializable)
  - class AFIT.Alm.Knapsack.Item
    - class AFIT.Alm.Knapsack.GeometricItem
  - class AFIT.Alm.Knapsack.ItemComparator (implements java.io.Serializable)
  - class AFIT.Alm.Knapsack.Reader.KnapSolve
  - class AFIT.Alm.Knapsack.Reader.KnapsackReader
  - class AFIT.Alm.Geometry.Matrix
    - class AFIT.Alm.Geometry.Matrix2d
  - class AFIT.Alm.Packing.Move
    - class AFIT.Alm.Packing.RotateMove
    - class AFIT.Alm.Packing.SwapMove
    - class AFIT.Alm.Packing.TranslateMove
  - class AFIT.Alm.Packing.MoveSet (implements java.io.Serializable)
  - class AFIT.Alm.Knapsack.MultidimensionalKnapsack
    - class AFIT.Alm.Knapsack.ReactiveKnapsack
      - class AFIT.Alm.Knapsack.GeometricKnapsack
  - class AFIT.Alm.Packing.ObjectiveFunction (implements java.io.Serializable)
  - class AFIT.Alm.Packing.Params

- class AFIT.Alm.triangulate.PointT
- class AFIT.Alm.Knapsack.Pointer (implements AFIT.Alm.Sort.Comparable)
- class AFIT.Alm.Knapsack.QuantityPredicate (implements java.io.Serializable)
- class AFIT.Alm.Knapsack.RTSParameters (implements java.io.Serializable)
- class AFIT.Alm.Packing.Section
- class AFIT.Alm.Knapsack.Slave (implements java.io.Serializable)
- class AFIT.Alm.Packing.Tabu
- class java.lang.Thread (implements java.lang.Runnable)
  - class AFIT.Alm.Packing.SearchThread
  - class AFIT.Alm.Packing.SearchViewer
- class AFIT.Alm.triangulate.Triangle
- class AFIT.Alm.triangulate.TriangulatePolygon
- class AFIT.Alm.Knapsack.UnitAlm (implements java.io.Serializable)
- class AFIT.Alm.Geometry.Vert2d
- class AFIT.Alm.Packing.bestMove (implements java.io.Serializable)

Index of all Fields and Methods

A

Aircraft(double[], double[], int, double, double, double). Constructor for class AFIT.Alm.Packing.Aircraft
allSelected(). Method in class AFIT.Alm.Knapsack.Item

B

BalancedContainer(double[], double[], int, double, double). Constructor for class AFIT.Alm.Packing.BalancedContainer
Instantiates a new Balanced Container

bestMove(). Constructor for class AFIT.Alm.Packing.bestMove
bestMove(). Method in class AFIT.Alm.Packing.MoveSet
Move the item by an absolute best Move.

C

C17(). Constructor for class AFIT.Alm.Packing.C17
Instantiates a C17 aircraft
calculateBounds(). Method in interface AFIT.Alm.Packing.Cargo
calculateBounds(). Method in class AFIT.Alm.Packing.Cargo2d
This method calculates the two dimensional bounding box of the cargo Item.
calculateBounds(). Method in class AFIT.Alm.Packing.Container
Calculates the bounding box of the container and updates the *width* and *height*
CandidateListStrategy(int). Constructor for class AFIT.Alm.Packing.CandidateListStrategy
Constructs the class that encapsulates the candidate list strategy For move sets less than 100, the minimum of (5,move set size) is used for the block size.
CandidateListStrategy(int, int). Constructor for class

AFIT.Alm.Packing.CandidateListStrategy
Constructs the class that encapsulates the candidate list strategy
Cargo2d(Cargo2d). Constructor for class AFIT.Alm.Packing.Cargo2d
Instantiates a new *Cargo2d* object with the same parameters as *c*

Cargo2d(double[], double[], int). Constructor for class AFIT.Alm.Packing.Cargo2d
Instantitiates an new *Cargo2d* item.
cargoCGLocationX(Cargo[], double). Static method in class AFIT.Alm.Packing.BalancedContainer
Determines the center of gravity location on the x axis of this container with array of *Cargo c* in the current packing pattern

cgLocationX(Cargo[]). Method in class AFIT.Alm.Packing.BalancedContainer
Determines the center of gravity location on the x axis of this container with array of *Cargo c* in the current packing pattern

checkTabu(). Method in class AFIT.Alm.Packing.Move
Checks to see if the move is tabu, after three calls to this method Tabu status is removed
clearTabu(). Method in class AFIT.Alm.Packing.Move
Remove from the Tabu status
clone(). Method in class AFIT.Alm.Knapsack.GeometricItem
clone(). Method in class AFIT.Alm.Knapsack.Item
clone(). Method in class AFIT.Alm.Knapsack.ItemOrderedSet
clone(). Method in class AFIT.Alm.Packing.Vehicle
compareTo(Comparable). Method in class AFIT.Alm.Knapsack.Item
compareTo(Comparable). Method in class AFIT.Alm.Knapsack.Pointer
Container(double, double, double, double). Constructor for class AFIT.Alm.Packing.Container
Instantiates a new rectangular shaped two dimensional `Container` with the upper left hand corner at point*x,y* with dimensions*width* and *height*
Container(double[], double[], int). Constructor for class AFIT.Alm.Packing.Container
Constructs a new polyon shaped Container with cordinates (`xPoints`, `yPoints`) The container must be convex or the *protrusion* mehtod will not work correctly.

D

decreaseQuantity(int). Method in class AFIT.Alm.Knapsack.ID
decreaseQuantity(int). Method in class AFIT.Alm.Knapsack.Item
doubleValue(String). Static method in class AFIT.Alm.Knapsack.Reader.EquipmentReader

draw(). Method in class AFIT.Alm.Packing.SearchViewer

E

equals(Object). Method in class AFIT.Alm.Knapsack.ID
equals(Object). Method in class AFIT.Alm.Knapsack.Item
equals(Object). Method in class AFIT.Alm.Geometry.Vert2d
Determines whether two vertices are equal.
EquipmentAlm(). Constructor for class AFIT.Alm.Knapsack.EquipmentAlm
EquipmentReader(). Constructor for class AFIT.Alm.Knapsack.Reader.EquipmentReader
execute(Object). Method in class AFIT.Alm.Knapsack.QuantityPredicate
execute(Object, Object). Method in class AFIT.Alm.Knapsack.ItemComparator
extentsOverlap(Cargo). Method in class AFIT.Alm.Packing.Cargo2d
Return true if the bounding box overlaps *Cargo* item *c*.

F

feasible(). Method in class AFIT.Alm.Packing.ObjectiveFunction
Returns true if the current packing pattern is feasible
feasible(). Method in class AFIT.Alm.Packing.Tabu

G

GeometricItem(double, double[], int, double, double). Constructor for class AFIT.Alm.Knapsack.GeometricItem
GeometricItem(GeometricItem). Constructor for class AFIT.Alm.Knapsack.GeometricItem
GeometricKnapsack(RTSParameters, ItemOrderedSet, double[], Aircraft). Constructor for class AFIT.Alm.Knapsack.GeometricKnapsack
Geometry2d(). Constructor for class AFIT.Alm.Geometry.Geometry2d
getArea(). Method in interface AFIT.Alm.Packing.Cargo
getArea(). Method in class AFIT.Alm.Packing.Cargo2d
Get the area of the polygon
getBestSet(). Method in class AFIT.Alm.Knapsack.MultidimensionalKnapsack
getBestTime(). Method in class AFIT.Alm.Knapsack.MultidimensionalKnapsack

getBestValue(). Method in class AFIT.Alm.Knapsack.MultidimensionalKnapsack
getbestValue(). Method in class AFIT.Alm.Packing.Tabu
getCentroidX(). Method in interface AFIT.Alm.Packing.Cargo
getCentroidX(). Method in class AFIT.Alm.Packing.Cargo2d
Get the x cordinate location of the centroid
getCentroidY(). Method in interface AFIT.Alm.Packing.Cargo
getCentroidY(). Method in class AFIT.Alm.Packing.Cargo2d
Get the x cordinate location of the centroid
getCpuTime(). Method in class AFIT.Alm.Knapsack.MultidimensionalKnapsack
getcurrentValue(). Method in class AFIT.Alm.Packing.Tabu
getEquipment(). Method in class AFIT.Alm.Knapsack.UnitAlm
getGeometricItem(). Method in class AFIT.Alm.Knapsack.EquipmentAlm
getGeometricItems(). Method in class AFIT.Alm.Knapsack.GroupAlm
getID(). Method in class AFIT.Alm.Knapsack.EquipmentAlm
getId(). Method in class AFIT.Alm.Knapsack.EquipmentAlm
getID(). Method in class AFIT.Alm.Knapsack.GroupAlm
getID(). Method in class AFIT.Alm.Knapsack.ID
getID(). Method in class AFIT.Alm.Knapsack.UnitAlm
getIntValueID(). Method in class AFIT.Alm.Knapsack.ID
getIntX(). Method in interface AFIT.Alm.Packing.Cargo
getIntX(). Method in class AFIT.Alm.Packing.Cargo2d
Returns an *int* array of the x cordinates of the vertices
getIntY(). Method in interface AFIT.Alm.Packing.Cargo
getIntY(). Method in class AFIT.Alm.Packing.Cargo2d
Returns an *int* array of the y cordinates of the vertices
getItem(). Method in class AFIT.Alm.Knapsack.EquipmentAlm
getItems(). Method in class AFIT.Alm.Knapsack.GroupAlm
getIterations(). Method in class AFIT.Alm.Knapsack.MultidimensionalKnapsack
getLength(). Method in class AFIT.Alm.Knapsack.EquipmentAlm
getLoadedWeight(). Method in class AFIT.Alm.Knapsack.EquipmentAlm
getLocationX(). Method in class AFIT.Alm.Geometry.Vert2d
getLocationY(). Method in class AFIT.Alm.Geometry.Vert2d
getMaxAcl(). Method in class AFIT.Alm.Packing.Aircraft
getMaxX(). Method in interface AFIT.Alm.Packing.Cargo
getMaxX(). Method in class AFIT.Alm.Packing.Cargo2d
getMaxY(). Method in interface AFIT.Alm.Packing.Cargo
getMaxY(). Method in class AFIT.Alm.Packing.Cargo2d
getMinX(). Method in interface AFIT.Alm.Packing.Cargo
getMinX(). Method in class AFIT.Alm.Packing.Cargo2d
getMinY(). Method in interface AFIT.Alm.Packing.Cargo
getMinY(). Method in class AFIT.Alm.Packing.Cargo2d
getName(). Method in class AFIT.Alm.Knapsack.EquipmentAlm
getName(). Method in class AFIT.Alm.Knapsack.GroupAlm

getName(). Method in class AFIT.Alm.Knapsack.UnitAlm
getNextBROS(). Method in class AFIT.Alm.Packing.CandidateListStrategy
Returns the next move to make during an Improving phase based on a Block Random Order Scan
getNextFROS(). Method in class AFIT.Alm.Packing.CandidateListStrategy
Returns the next move to make during a Mixed phase Based on a Full Random Order Scan, until the move set is exhausted, then reverts to the Block Random Order Scan
getNPoints(). Method in interface AFIT.Alm.Packing.Cargo
getNPoints(). Method in class AFIT.Alm.Packing.Cargo2d
Get the number of vertices in the polygon that represents the *Cargo* item
getNpoints(). Method in class AFIT.Alm.Packing.Container
Returns the array of number of points or vertices that make up the container
getNumberOfEquipmentTypes(). Method in class AFIT.Alm.Knapsack.UnitAlm
getNumberOfItems(). Method in class AFIT.Alm.Knapsack.GroupAlm
getNumberOfTriangles(). Method in class AFIT.Alm.triangulate.TriangulatePolygon
Returns the number of triangle objects in the triangulated polygon
getNumberOfUnits(). Method in class AFIT.Alm.Knapsack.GroupAlm
getNumberSelected(). Method in class AFIT.Alm.Knapsack.Item
getNumTri(). Method in class AFIT.Alm.Packing.Cargo2d
Get the number of triangles.
getOverlap(). Method in class AFIT.Alm.Packing.Vehicle
The overlap of the Vehicle with other *Cargo* items
getProtrusion(Cargo). Method in class AFIT.Alm.Packing.Container
The protrusion distance is calculated using P = Px + Py where Px is the distance in the x direction that c is from the centroid of the container and Py is the distance in the y direction c from the centroid of the container.
getQuantity(). Method in class AFIT.Alm.Packing.Aircraft
getQuantity(). Method in class AFIT.Alm.Knapsack.GroupAlm
getQuantity(). Method in class AFIT.Alm.Knapsack.ID
getQuantity(). Method in class AFIT.Alm.Knapsack.Item
getTriangles(). Method in class AFIT.Alm.Packing.Cargo2d
Get the *Triangle* array of this *Cargo2d*
getTriangles(). Method in class AFIT.Alm.triangulate.TriangulatePolygon
This returns an array of Triangles that contains the triangle vertice numbers.
getUnPackedSet(). Method in class AFIT.Alm.Knapsack.MultidimensionalKnapsack
getValue(). Method in class AFIT.Alm.Packing.Move
getVehicle(). Method in class AFIT.Alm.Knapsack.GeometricItem
getVertex0(). Method in class AFIT.Alm.triangulate.Triangle
getVertex1(). Method in class AFIT.Alm.triangulate.Triangle
getVertex2(). Method in class AFIT.Alm.triangulate.Triangle
getWeight(). Method in interface AFIT.Alm.Packing.Cargo
getWeight(). Method in class AFIT.Alm.Packing.Cargo2d
Gets the weight of this cargo item

getWidth(). Method in class AFIT.Alm.Knapsack.EquipmentAlm
getXLocal(). Method in interface AFIT.Alm.Packing.Cargo
getXLocal(). Method in class AFIT.Alm.Packing.Cargo2d
Get the x cordinates of the local space
getXpoint(int). Method in interface AFIT.Alm.Packing.Cargo
getXpoint(int). Method in class AFIT.Alm.Packing.Cargo2d
Get the x cordinate of the vertice *index*
getXpoints(). Method in interface AFIT.Alm.Packing.Cargo
getXpoints(). Method in class AFIT.Alm.Packing.Cargo2d
Get the x cordinates of the vertices
getXpoints(). Method in class AFIT.Alm.Packing.Container
Returns the array of xPoints that make up the container
getYaw(). Method in interface AFIT.Alm.Packing.Cargo
getYaw(). Method in class AFIT.Alm.Packing.Cargo2d
Returns the yaw in degrees
getYLocal(). Method in interface AFIT.Alm.Packing.Cargo
getYLocal(). Method in class AFIT.Alm.Packing.Cargo2d
Get the y cordinates of the local space
getYpoint(int). Method in interface AFIT.Alm.Packing.Cargo
getYpoint(int). Method in class AFIT.Alm.Packing.Cargo2d
Get the y cordinate of the vertice *index*
getYpoints(). Method in interface AFIT.Alm.Packing.Cargo
getYpoints(). Method in class AFIT.Alm.Packing.Cargo2d
Get the y cordinates of the vertices
getYpoints(). Method in class AFIT.Alm.Packing.Container
Returns the array of yPoints that make up the container
GroupAlm(). Constructor for class AFIT.Alm.Knapsack.GroupAlm

H

hashCode(). Method in class AFIT.Alm.Knapsack.ID
hashCode(). Method in class AFIT.Alm.Knapsack.Item
hashCode(). Method in class AFIT.Alm.Knapsack.ItemOrderedSet
height(). Method in interface AFIT.Alm.Packing.Cargo
height(). Method in class AFIT.Alm.Packing.Cargo2d
The height of the bounding box of the cargo item
Helicopter(). Constructor for class AFIT.Alm.Packing.Helicopter
Instantiates a *Helicopter*

I

ID(int, int). Constructor for class AFIT.Alm.Knapsack.ID
improvingMove(). Method in class AFIT.Alm.Packing.MoveSet
    Move the Cargo item by a probalistic best move
improvingPhase(). Method in class AFIT.Alm.Packing.Tabu
increaseQuantity(int). Method in class AFIT.Alm.Knapsack.ID
increaseQuantity(int). Method in class AFIT.Alm.Knapsack.Item
initialize(). Method in class AFIT.Alm.Knapsack.Item
inside(double, double, double[], double[], int). Static method in class AFIT.Alm.Geometry.Geometry2d
intersectArea(Cargo2d). Method in class AFIT.Alm.Packing.Cargo2d
    The intersection area of *this* cargo item with another cargo item *c*
intersectArea(Cargo2d[]). Method in class AFIT.Alm.Packing.Cargo2d
    The intersection of this *Cargo* item with array of *Cargo* items *c*
intersectArea(Cargo2d[]). Method in class AFIT.Alm.Packing.Vehicle
    The intersectArea of the *Cargo* array with this *Vehicle*
intersectArea(Cargo[]). Method in interface AFIT.Alm.Packing.Cargo
intersectArea(Cargo[]). Method in class AFIT.Alm.Packing.Cargo2d
    The intersection area of *this* cargo item with an array of*Cargo* items *c*
intersectAreaAll(Cargo2d[]). Static method in class AFIT.Alm.Packing.Cargo2d
    The intersection area of all cargo items in array *c*
intersectAreaAll(Cargo[]). Method in interface AFIT.Alm.Packing.Cargo
intersectAreaAll(Cargo[]). Method in class AFIT.Alm.Packing.Cargo2d
    The intersection of this *Cargo* item with array of *Cargo* items *c*
intValue(String). Static method in class AFIT.Alm.Knapsack.Reader.EquipmentReader
ItemComparator(). Constructor for class AFIT.Alm.Knapsack.ItemComparator
ItemOrderedSet(). Constructor for class AFIT.Alm.Knapsack.ItemOrderedSet

---

K

KnapsackReader(). Constructor for class AFIT.Alm.Knapsack.Reader.KnapsackReader
KnapSolve(). Constructor for class AFIT.Alm.Knapsack.Reader.KnapSolve

---

L

length(). Method in class AFIT.Alm.Packing.Container
    Length of the bounding box

linesIntersect(double, double, double, double, double, double, double, double, Vert2d). Static method in class AFIT.Alm.Geometry.Geometry2d

---

M

main(String[]). Static method in class AFIT.Alm.Packing.CandidateListStrategy
test stub for the class
main(String[]). Static method in class AFIT.Alm.Packing.Container
test stub for the class
main(String[]). Static method in class AFIT.Alm.Knapsack.Reader.EquipmentReader
main(String[]). Static method in class AFIT.Alm.Geometry.Geometry2d
main(String[]). Static method in class AFIT.Alm.Knapsack.Reader.KnapsackReader
main(String[]). Static method in class AFIT.Alm.Knapsack.MultidimensionalKnapsack
main(String[]). Static method in class AFIT.Alm.Knapsack.ReactiveKnapsack
Matrix(). Constructor for class AFIT.Alm.Geometry.Matrix
Matrix2d(). Constructor for class AFIT.Alm.Geometry.Matrix2d
MatrixTransform(). Method in interface AFIT.Alm.Packing.Cargo
MatrixTransform(). Method in class AFIT.Alm.Packing.Cargo2d
Moves *Cargo2d* by its matrix.
maxy. Variable in class AFIT.Alm.Packing.Section
minY. Variable in class AFIT.Alm.Packing.Section
mixedMove(). Method in class AFIT.Alm.Packing.MoveSet
Makes a random swap or rotate move
mixedPhase(). Method in class AFIT.Alm.Packing.Tabu
Move(). Constructor for class AFIT.Alm.Packing.Move
move(). Method in class AFIT.Alm.Packing.Move
Move a *Cargo object to some destination*
move(). *Method in class AFIT.Alm.Packing.RotateMove*
Rotate a Cargo object theta degrees around the z axis
move(). *Method in class AFIT.Alm.Packing.SwapMove*
Swaps this item with another Item
move(). *Method in class AFIT.Alm.Packing.TranslateMove*
Moves the Cargo Item by xDis,yDis
moveSet. *Variable in class AFIT.Alm.Packing.Cargo2d*
The MoveSet for this cargo item
MoveSet(*Cargo, Cargo[], ObjectiveFunction, double, double*). *Constructor for class AFIT.Alm.Packing.MoveSet*
Constructs a new Cargo object.
MultidimensionalKnapsack(*double[], double[], double[][]*). *Constructor for class AFIT.Alm.Knapsack.MultidimensionalKnapsack*

MultidimensionalKnapsack(double[], double[], double[][], int, int, int). *Constructor for class AFIT.Alm.Knapsack.MultidimensionalKnapsack*
MultidimensionalKnapsack(ItemOrderedSet, double[]). *Constructor for class AFIT.Alm.Knapsack.MultidimensionalKnapsack*

N

newSwapItem(). *Method in class AFIT.Alm.Packing.SwapMove*
Generates a new Item to swap with this item
noneSelected(). *Method in class AFIT.Alm.Knapsack.Item*
numTri. *Variable in class AFIT.Alm.Packing.Cargo2d*

O

ObjectiveFunction(Cargo[], Aircraft). *Constructor for class AFIT.Alm.Packing.ObjectiveFunction*
Constructs a new ObjectiveFunction

objFunct. *Variable in class AFIT.Alm.Packing.Tabu*
objFunction(). *Method in class AFIT.Alm.Packing.ObjectiveFunction*
Evaluate the current Packing Pattern, this ignores the weights
objFunctionItem(Cargo). *Method in class AFIT.Alm.Packing.ObjectiveFunction*
Evaluate posistion of an item based on current posistion using weights
output(). *Method in class AFIT.Alm.Knapsack.Item*
output(PrintWriter, RTSParameters). *Static method in class AFIT.Alm.Knapsack.RTSParameters*
outputSet(PrintWriter, ItemOrderedSet). *Static method in class AFIT.Alm.Knapsack.ItemOrderedSet*

P

PackCanvas(Aircraft, Cargo[]). *Constructor for class AFIT.Alm.Packing.PackCanvas*
PackingCanvas(Aircraft, Cargo[]). *Constructor for class AFIT.Alm.Packing.PackingCanvas*
paint(Graphics). *Method in class AFIT.Alm.Packing.PackCanvas*
paint(Graphics). *Method in class AFIT.Alm.Packing.PackingCanvas*

***Params*()**. Constructor for class AFIT.Alm.Packing.Params*
***Pointer*(int, double)**. Constructor for class AFIT.Alm.Knapsack.Pointer*

Q

***QuantityPredicate*()**. Constructor for class AFIT.Alm.Knapsack.QuantityPredicate*

R

***ReactiveKnapsack*(RTSParameters, double[], double[], double[][])**. Constructor for class AFIT.Alm.Knapsack.ReactiveKnapsack*
***ReactiveKnapsack*(RTSParameters, ItemOrderedSet, double[])**. Constructor for class AFIT.Alm.Knapsack.ReactiveKnapsack*
***readEquipmentData*()**. Static method in class AFIT.Alm.Knapsack.Reader.EquipmentReader*
***readGroupData*()**. Static method in class AFIT.Alm.Knapsack.Reader.EquipmentReader*
***readUnitData*()**. Static method in class AFIT.Alm.Knapsack.Reader.EquipmentReader*
***remove*(Object)**. Method in class AFIT.Alm.Knapsack.ItemOrderedSet*
***resetNorms*()**. Method in class AFIT.Alm.Packing.ObjectiveFunction*
Set the norms back to a constant value
***rotate*(double)**. Method in interface AFIT.Alm.Packing.Cargo*
***rotate*(double)**. Method in class AFIT.Alm.Packing.Cargo2d*
This method rotates the Cargo item around centroidX and centroidY by theta degrees and then updates the bounding box.
***rotate*(double)**. Method in class AFIT.Alm.Geometry.Matrix*
***rotate*(double)**. Method in class AFIT.Alm.Geometry.Matrix2d*
***RotateMove*(Cargo, double)**. Constructor for class AFIT.Alm.Packing.RotateMove*
Constructs a new Rotate move for Cargo item that will rotate theta degrees around the z axis

***RTSParameters*()**. Constructor for class AFIT.Alm.Knapsack.RTSParameters*
***RTSParameters*(int, int, int, int)**. Constructor for class AFIT.Alm.Knapsack.RTSParameters*
***run*()**. Method in class AFIT.Alm.Packing.SearchThread*
Executes the packing search
***run*()**. Method in class AFIT.Alm.Packing.SearchViewer*

S

sameSign(double, double). Static method in class AFIT.Alm.Geometry.Geometry2d
    This method

    The method uses the following code:!((a >= 0.0d)^(b >= 0.0d)) to determine if a and b are the same sign.
SearchThread(Tabu, int, JCProgressMeter). Constructor for class AFIT.Alm.Packing.SearchThread
    Instantiates a new SearchThread

SearchViewer(Canvas, Aircraft, Cargo[], Params, FormattedTextField, FormattedTextField, FormattedTextField, FormattedTextField, FormattedTextField, FormattedTextField). Constructor for class AFIT.Alm.Packing.SearchViewer
Section(double, double, double). Constructor for class AFIT.Alm.Packing.Section
Section(double, double, double, String). Constructor for class AFIT.Alm.Packing.Section
SectionedAircraft(Section[], int, double, double, double). Constructor for class AFIT.Alm.Packing.SectionedAircraft
    Instantiates a sectioned Aircraft.
setCentroid(double, double). Method in interface AFIT.Alm.Packing.Cargo
setCentroid(double, double). Method in class AFIT.Alm.Packing.Cargo2d
    Set the centroid to the location x, and location y
setCentroid(Vert2d). Method in class AFIT.Alm.Packing.Cargo2d
    Set the centroid of the cargo item to the vertice c
setEmptyWeight(double). Method in class AFIT.Alm.Knapsack.EquipmentAlm
setEquipment(ID[]). Method in class AFIT.Alm.Knapsack.UnitAlm
setHeight(double). Method in class AFIT.Alm.Knapsack.EquipmentAlm
setId(int). Method in class AFIT.Alm.Knapsack.EquipmentAlm
setId(int). Method in class AFIT.Alm.Knapsack.GroupAlm
setId(int). Method in class AFIT.Alm.Knapsack.UnitAlm
setItem(Item). Method in class AFIT.Alm.Knapsack.Item
setLength(double). Method in class AFIT.Alm.Knapsack.EquipmentAlm
setLoadedWeight(double). Method in class AFIT.Alm.Knapsack.EquipmentAlm
setLocation(double, double). Method in class AFIT.Alm.Geometry.Vert2d
setMatrixRotate(double). Method in interface AFIT.Alm.Packing.Cargo
setMatrixRotate(double). Method in class AFIT.Alm.Packing.Cargo2d
    Rotate this Cargo2d Matrix by theta
setMatrixTranslate(double, double). Method in interface AFIT.Alm.Packing.Cargo
setMatrixTranslate(double, double). Method in class AFIT.Alm.Packing.Cargo2d
    Translate this Cargo2d Matrix by x in the x direction and by y in the y direction
setMatrixUnit(). Method in interface AFIT.Alm.Packing.Cargo
setMatrixUnit(). Method in class AFIT.Alm.Packing.Cargo2d
    Set the transform matrix of this cargo item to the identity matrix
setMaxGrossWeight(double). Method in class AFIT.Alm.Knapsack.EquipmentAlm
setMaxOuterSpan(int). Method in class AFIT.Alm.Knapsack.MultidimensionalKnapsack

setMoveSet(Cargo[], ObjectiveFunction, double, double). Method in class AFIT.Alm.Packing.Cargo2d
setName(String). Method in class AFIT.Alm.Knapsack.EquipmentAlm
setName(String). Method in class AFIT.Alm.Knapsack.GroupAlm
setName(String). Method in class AFIT.Alm.Knapsack.UnitAlm
setNomenclature(String). Method in class AFIT.Alm.Knapsack.EquipmentAlm
setNomenclature(String). Method in class AFIT.Alm.Knapsack.GroupAlm
setNomenclature(String). Method in class AFIT.Alm.Knapsack.UnitAlm
setNumberOfEquipmentTypes(int). Method in class AFIT.Alm.Knapsack.UnitAlm
setNumberOfPassengers(int). Method in class AFIT.Alm.Knapsack.UnitAlm
setNumberOfUnits(int). Method in class AFIT.Alm.Knapsack.GroupAlm
setProfitPerLBS(double). Method in class AFIT.Alm.Knapsack.EquipmentAlm
setQuantity(int). Method in class AFIT.Alm.Packing.Aircraft
setQuantity(int). Method in class AFIT.Alm.Knapsack.ID
setQuantity(int). Method in class AFIT.Alm.Knapsack.Item
setTabu(). Method in class AFIT.Alm.Packing.Move
Place on Tabu Status
setToBestFound(). Method in class AFIT.Alm.Packing.SearchThread
Sets the packing pattern to best found patter
setToBestFound(). Method in class AFIT.Alm.Packing.Tabu
setUnits(ID[]). Method in class AFIT.Alm.Knapsack.GroupAlm
setValue(double). Method in class AFIT.Alm.Packing.Move
Sets the value of this move
setWeight(double). Method in class AFIT.Alm.Packing.Cargo2d
Sets the weight of this cargo item
setWeightAccompanySupplies(double). Method in class AFIT.Alm.Knapsack.UnitAlm
setWeightAmmo(double). Method in class AFIT.Alm.Knapsack.UnitAlm
setWeightNonMobilEquipment(double). Method in class AFIT.Alm.Knapsack.UnitAlm
setWeightNonMobilPallets(double). Method in class AFIT.Alm.Knapsack.UnitAlm
setWeights(double, double, double, double). Method in class AFIT.Alm.Packing.ObjectiveFunction
Set the weights for Ovelap penalty, Bounding Box Penalty,Protrision Penalty, and Centr of Gravity penalties
setWidth(double). Method in class AFIT.Alm.Knapsack.EquipmentAlm
setYaw(double). Method in interface AFIT.Alm.Packing.Cargo
setYaw(double). Method in class AFIT.Alm.Packing.Cargo2d
Set the yaw in degrees
Slave(). Constructor for class AFIT.Alm.Knapsack.Slave
solve(). Method in class AFIT.Alm.Knapsack.MultidimensionalKnapsack
solve(Aircraft, ItemOrderedSet, int, int, int). Static method in class AFIT.Alm.Knapsack.Slave
solve(int, PrintWriter, int, double[], double[], double[][]). Static method in class AFIT.Alm.Knapsack.Reader.KnapSolve
solveKnapsack(File, File, RTSParameters). Static method in class AFIT.Alm.Knapsack.Reader.KnapsackReader
solveRTS(int, PrintWriter, RTSParameters, double[], double[], double[][]). Static method in class AFIT.Alm.Knapsack.Reader.KnapSolve
swap(Cargo). Method in interface AFIT.Alm.Packing.Cargo
swap(Cargo). Method in class AFIT.Alm.Packing.Cargo2d
This method swaps the location of this Cargo item to the location

of Cargo item c based on centroid position.
SwapMove(Cargo, Cargo[], int, int). Constructor for class AFIT.Alm.Packing.SwapMove
Constructs a new Swap move for Cargo item that will swap item with another item between in the array cargoArray between the index of minIndex and maxIndex

---

T

Tabu(Aircraft, Cargo[], int, int). Constructor for class AFIT.Alm.Packing.Tabu
transform(double[], double[], double[], double[], int). Method in class AFIT.Alm.Geometry.Matrix
transform(double[], double[], double[], double[], int). Method in class AFIT.Alm.Geometry.Matrix2d
This transforms the arrays xcord and ycord by the transformation matrix and outputs into tx and ty

transform(Vert2d[], Vert2d[]). Method in class AFIT.Alm.Geometry.Matrix2d
translate(double, double). Method in interface AFIT.Alm.Packing.Cargo
translate(double, double). Method in class AFIT.Alm.Packing.Cargo2d
This method moves the Cargo item by deltaX in the x direction and deltaY in the y direction

translate(double, double). Method in class AFIT.Alm.Geometry.Matrix
translate(double, double). Method in class AFIT.Alm.Geometry.Matrix2d
TranslateMove(Cargo, double, double). Constructor for class AFIT.Alm.Packing.TranslateMove
Constructs a new TranslateMove or Cargo item that will translate the item xDis in the x direction and yDis in the y direction

Triangle(int[]). Constructor for class AFIT.Alm.triangulate.Triangle
Instantiates a Triangle

triangleIntersect(Triangle, double[], double[], double[], double[]). Method in class AFIT.Alm.triangulate.Triangle
Determines if this triangle intersects another triangle using the methods described in Theodoractos and Grimsley's article The optimal packing of arbitrarily-shaped polygons using simulated annealing and polynomial-time cooling schedules in Computer Methods in applied mechanics and engineering

TriangulatePolygon(int, int[], double[][]). Constructor for class AFIT.Alm.triangulate.TriangulatePolygon
This instatiates the Triangulate Polygon Class.

U

unit(). Method in class AFIT.Alm.Geometry.Matrix
unit(). Method in class AFIT.Alm.Geometry.Matrix2d
UnitAlm(). Constructor for class AFIT.Alm.Knapsack.UnitAlm
unmove(). Method in class AFIT.Alm.Packing.Move
    Undo the last move made by a Cargo object
unmove(). Method in class AFIT.Alm.Packing.RotateMove
    Rotate a Cargo object negative theta degrees around the z axis
unmove(). Method in class AFIT.Alm.Packing.SwapMove
    Undoes the swaps between this item with another Item
unmove(). Method in class AFIT.Alm.Packing.TranslateMove
    Moves the Cargo Item by -xDis,-yDis
updateExtents(). Method in class AFIT.Alm.Packing.Cargo2d
    Update the cordinates of the Traingle array to the current location
updateExtents(double[], double[]). Method in class AFIT.Alm.triangulate.Triangle
    Updates the actual position of the bounding box of the Triangle.
upDateQuantityToNotSelected(). Method in class AFIT.Alm.Knapsack.Item
upDateQuantityToSelected(). Method in class AFIT.Alm.Knapsack.Item

---

V

Vehicle(double, double, double, double). Constructor for class AFIT.Alm.Packing.Vehicle
    Constructs a new vehicle with the upper left hand corner at point x,y and with width and height of variables with the same name.
Vehicle(Vehicle). Constructor for class AFIT.Alm.Packing.Vehicle
    Constructs a vehicle with the same dimensions of v
Vert2d(). Constructor for class AFIT.Alm.Geometry.Vert2d
Vert2d(double, double). Constructor for class AFIT.Alm.Geometry.Vert2d
    Constructs and initializes a vertice at the specified (x, y) location in the coordinate space.
Vert2d(Vert2d). Constructor for class AFIT.Alm.Geometry.Vert2d

---

W

width(). Method in interface AFIT.Alm.Packing.Cargo
width(). Method in class AFIT.Alm.Packing.Cargo2d
    The width of the bounding box of the cargo item
width(). Method in class AFIT.Alm.Packing.Container
    Width of the bounding box

X

x. *Variable in class AFIT.Alm.triangulate.PointT*
x cordinate
x. *Variable in class AFIT.Alm.Geometry.Vert2d*
The x coordinate.

Y

y. *Variable in class AFIT.Alm.triangulate.PointT*
y cordinate
y. *Variable in class AFIT.Alm.Geometry.Vert2d*
The y coordinate.

package AFIT.Alm.Geometry

Class Index

- Geometry2d
- Matrix
- Matrix2d
- Vert2d

Class AFIT.Alm.Geometry.Geometry2d

```
java.lang.Object
   |
   +----AFIT.Alm.Geometry.Geometry2d
```

public abstract class **Geometry2d**
extends Object

Constructor Index

- **Geometry2d**()

Method Index

- **inside**(double, double, double[], double[], int)
- **linesIntersect**(double, double, double, double, double, double, double, double, Vert2d)
- **main**(String[])
- **sameSign**(double, double)
  This method

  The method uses the following code: `!( (a >= 0.0d)^(b >= 0.0d) )` to determine if a and b are the same sign.

Constructors

Geometry2d

```
public Geometry2d()
```

Methods

sameSign

```
public static final boolean sameSign(double a,
                                     double b)
```

This method

The method uses the following code:!((a >= 0.0d)^(b >= 0.0d)) to determine if a and b are the same sign.

Parameters:
a - a first number to compare
b - b second number to compare

Returns:
returns true if a and b are both the same sign false other wise.

linesIntersect

```
public static final int linesIntersect(double x1,
                                       double y1,
                                       double x2,
                                       double y2,
                                       double x3,
                                       double y3,
                                       double x4,
                                       double y4,
                                       Vert2d v)
```

inside

```
public static final boolean inside(double x,
                                   double y,
                                   double xpoints[],
                                   double ypoints[],
                                   int npoints)
```

main

```
public static void main(String args[])
```

Class AFIT.Alm.Geometry.Matrix

```
java.lang.Object
   |
   +----AFIT.Alm.Geometry.Matrix
```

---

public abstract class **Matrix**
extends Object

Constructor Index

- **Matrix**()

Method Index

- **rotate**(double)
- **transform**(double[], double[], double[], double[], int)
- **translate**(double, double)
- **unit**()

Constructors

Matrix

```
public Matrix()
```

Methods

unit

```
public abstract void unit()
```

translate

```
public abstract void translate(double dx,
                               double dy)
```

rotate

```
public abstract void rotate(double theta)
```

transform

```
public abstract void transform(double x[],
                               double y[],
                               double tx[],
                               double ty[],
                               int nvert)
```

Class AFIT.Alm.Geometry.Matrix2d

```
java.lang.Object
   |
   +----AFIT.Alm.Geometry.Matrix
           |
           +----AFIT.Alm.Geometry.Matrix2d
```

public class **Matrix2d**
extends Matrix

Constructor Index

- **Matrix2d**()

Method Index

- **rotate**(double)
- **transform**(double[], double[], double[], double[], int)
  This transforms the arrays `xcord` and `ycord` by the transformation matrix and outputs into `tx` and `ty`
- **transform**(Vert2d[], Vert2d[])
- **translate**(double, double)
- **unit**()

Constructors

Matrix2d

```
public Matrix2d()
```

Methods

● translate

```
public void translate(double dx,
                      double dy)
```

Overrides:
translate in class Matrix

● rotate

```
public void rotate(double theta)
```

Overrides:
rotate in class Matrix

● unit

```
public void unit()
```

Overrides:
unit in class Matrix

● transform

```
public void transform(double xcord[],
                      double ycord[],
                      double tx[],
                      double ty[],
                      int nvert)
```

This transforms the arrays `xcord` and `ycord` by the transformation matrix and outputs into `tx` and `ty`

Parameters:
- xcord - Input x cordinates
- ycord - Input y cordinates
- tx - Output x cordinates
- ty - Output y cordiantes
- nvert - Number of Vertices in arrays

Overrides:
transform in class Matrix

● transform

```
public void transform(Vert2d in[],
                      Vert2d out[])
```

Class AFIT.Alm.Geometry.Vert2d

```
java.lang.Object
   |
   +----AFIT.Alm.Geometry.Vert2d
```

---

public class **Vert2d**
extends Object

Variable Index

- **x**
  The *x* coordinate.
- **y**
  The *y* coordinate.

Constructor Index

- **Vert2d**()
- **Vert2d**(double, double)
  Constructs and initializes a vertice at the specified (*x*, *y*) location in the coordinate space.
- **Vert2d**(Vert2d)

Method Index

- **equals**(Object)
  Determines whether two vertices are equal.
- **getLocationX**()
- **getLocationY**()
- **setLocation**(double, double)

Variables

x

```
public double x
```

The *x* coordinate.

y

```
public double y
```

The *y* coordinate.

Constructors

Vert2d

```
public Vert2d(double x,
              double y)
```

Constructs and initializes a vertice at the specified (*x*, *y*) location in the coordinate space.

Parameters:

x - the *x* coordinate.
y - the *y* coordinate.

Vert2d

```
public Vert2d(Vert2d v)
```

Vert2d

```
public Vert2d()
```

Methods

setLocation

```
public void setLocation(double x,
                        double y)
```

getLocationX

```
public double getLocationX()
```

getLocationY

```
public double getLocationY()
```

equals

```
public boolean equals(Object obj)
```

Determines whether two vertices are equal. Two instances of `Vert2d` are equal if the values of their `x` and `y` member fields, representing their position in the coordinate space, are the same.

Parameters:
obj - an object to be compared with this point.

Returns:
`true` if the object to be compared is an instance of `Point` and has the same values; `false` otherwise.

Overrides:
equals in class Object

---

package AFIT.Alm.Knapsack

Class Index

- EquipmentAlm
- GeometricItem
- GeometricKnapsack
- GroupAlm
- ID
- Item
- ItemComparator
- ItemOrderedSet
- MultidimensionalKnapsack
- Pointer
- QuantityPredicate
- RTSParameters
- ReactiveKnapsack
- Slave
- UnitAlm

Class AFIT.Alm.Knapsack.EquipmentAlm

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.EquipmentAlm
```

---

public class **EquipmentAlm**
extends Object
implements Serializable

Constructor Index

- **EquipmentAlm**()

Method Index

- **getGeometricItem**()
- **getID**()
- **getId**()
- **getItem**()
- **getLength**()
- **getLoadedWeight**()
- **getName**()
- **getWidth**()
- **setEmptyWeight**(double)
- **setHeight**(double)
- **setId**(int)
- **setLength**(double)
- **setLoadedWeight**(double)
- **setMaxGrossWeight**(double)
- **setName**(String)
- **setNomenclature**(String)

- setProfitPerLBS(double)
- setWidth(double)

Constructors

EquipmentAlm

```
public EquipmentAlm()
```

Methods

getItem

```
public final Item getItem()
```

getGeometricItem

```
public final GeometricItem getGeometricItem()
```

getID

```
public final Integer getID()
```

setName

```
public final void setName(String n)
```

setId

```
public final void setId(int i)
```

setLength

```
public final void setLength(double l)
```

setWidth

```
public final void setWidth(double w)
```

setHeight

```
public final void setHeight(double h)
```

setLoadedWeight

```
public final void setLoadedWeight(double w)
```

setNomenclature

```
public final void setNomenclature(String s)
```

setMaxGrossWeight

```
public final void setMaxGrossWeight(double w)
```

setEmptyWeight

```
public final void setEmptyWeight(double w)
```

setProfitPerLBS

```
public final void setProfitPerLBS(double p)
```

getName

```
public final String getName()
```

getId

```
public final int getId()
```

getLength

```
public final double getLength()
```

getWidth

```
public final double getWidth()
```

getLoadedWeight

```
public final double getLoadedWeight()
```

Class AFIT.Alm.Knapsack.GeometricItem

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.Item
           |
           +----AFIT.Alm.Knapsack.GeometricItem
```

public class **GeometricItem**
extends Item

Constructor Index

- **GeometricItem**(double, double[], int, double, double)
- **GeometricItem**(GeometricItem)

Method Index

- **clone**()
- **getVehicle**()

Constructors

GeometricItem

```
 public GeometricItem(double profit,
                      double constraint[],
                      int id,
                      double length,
                      double width)
```

GeometricItem

```
public GeometricItem(GeometricItem item)
```

Methods

clone

```
public final Object clone()
```

Overrides:
clone in class Item

getVehicle

```
public final Vehicle getVehicle()
```

Class AFIT.Alm.Knapsack.GroupAlm

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.GroupAlm
```

public class **GroupAlm**
extends Object
implements Serializable

Constructor Index

- GroupAlm()

Method Index

- getGeometricItems()
- getID()
- getItems()
- getName()
- getNumberOfItems()
- getNumberOfUnits()
- getQuantity()
- setId(int)
- setName(String)
- setNomenclature(String)
- setNumberOfUnits(int)
- setUnits(ID[])

Constructors

GroupAlm

```
public GroupAlm()
```

Methods

getItems

```
public final Item[] getItems()
```

getGeometricItems

```
public final GeometricItem[] getGeometricItems()
```

getID

```
public final Integer getID()
```

getNumberOfItems

```
public final int getNumberOfItems()
```

getQuantity

```
public final int getQuantity()
```

getName

```
public final String getName()
```

getNumberOfUnits

```
public final int getNumberOfUnits()
```

setName

```
public final void setName(String s)
```

setId

```
public final void setId(int i)
```

setNomenclature

```
public final void setNomenclature(String s)
```

setNumberOfUnits

```
public final void setNumberOfUnits(int u)
```

● setUnits

```
public final void setUnits(ID u[])
```

Class AFIT.Alm.Knapsack.ID

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.ID
```

public class **ID**
extends Object
implements Serializable

Constructor Index

- **ID**(int, int)

Method Index

- **decreaseQuantity**(int)
- **equals**(Object)
- **getID**()
- **getIntValueID**()
- **getQuantity**()
- **hashCode**()
- **increaseQuantity**(int)
- **setQuantity**(int)

Constructors

ID

```
 public ID(int i,
           int q)
```

Methods

hashCode

```
public final int hashCode()
```

Overrides:
hashCode in class Object

equals

```
public final boolean equals(Object object)
```

Overrides:
equals in class Object

getID

```
public final Integer getID()
```

getIntValueID

```
public final int getIntValueID()
```

getQuantity

```
public final int getQuantity()
```

setQuantity

```
public final void setQuantity(int q)
```

increaseQuantity

```
public final void increaseQuantity(int q)
```

decreaseQuantity

```
public final void decreaseQuantity(int q)
```

Class AFIT.Alm.Knapsack.Item

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.Item
```

---

public class **Item**
extends Object

Method Index

- **allSelected**()
- **clone**()
- **compareTo**(Comparable)
- **decreaseQuantity**(int)
- **equals**(Object)
- **getNumberSelected**()
- **getQuantity**()
- **hashCode**()
- **increaseQuantity**(int)
- **initialize**()
- **noneSelected**()
- **output**()
- **setItem**(Item)
- **setQuantity**(int)
- **upDateQuantityToNotSelected**()
- **upDateQuantityToSelected**()

Methods

clone

```
public Object clone()
```

Overrides:

clone in class Object

getNumberSelected

```
public final int getNumberSelected()
```

setItem

```
public final void setItem(Item i)
```

initialize

```
public final void initialize()
```

upDateQuantityToNotSelected

```
public final void upDateQuantityToNotSelected()
```

upDateQuantityToSelected

```
public final void upDateQuantityToSelected()
```

getQuantity

```
public final int getQuantity()
```

allSelected

```
public final boolean allSelected()
```

noneSelected

```
public final boolean noneSelected()
```

setQuantity

```
public final void setQuantity(int q)
```

increaseQuantity

```
public final void increaseQuantity(int q)
```

decreaseQuantity

```
public final void decreaseQuantity(int q)
```

hashCode

```
public final int hashCode()
```

Overrides:

hashCode in class Object

equals

```
public boolean equals(Object object)
```

Overrides:

equals in class Object

compareTo

```
public int compareTo(Comparable b)
```

output

```
public final String output()
```

Class AFIT.Alm.Knapsack.ItemComparator

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.ItemComparator
```

public class **ItemComparator**
extends Object
implements Serializable

Constructor Index

- **ItemComparator**()

Method Index

- **execute**(Object, Object)

Constructors

ItemComparator

```
public ItemComparator()
```

Methods

execute

```
public final boolean execute(Object first,
                             Object second)
```

Class AFIT.Alm.Knapsack.ItemOrderedSet

AFIT.Alm.Knapsack.ItemOrderedSet

public class **ItemOrderedSet**

Constructor Index

- **ItemOrderedSet**()

Method Index

- **clone**()
- **hashCode**()
- **outputSet**(PrintWriter, ItemOrderedSet)
- **remove**(Object)

Constructors

ItemOrderedSet

```
public ItemOrderedSet()
```

Methods

clone

```
public synchronized Object clone()
```

outputSet

```
public static final void outputSet(PrintWriter out,
                                   ItemOrderedSet set)
```

remove

```
public final int remove(Object object)
```

hashCode

```
public final int hashCode()
```

Class AFIT.Alm.Knapsack.MultidimensionalKn

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.MultidimensionalKnapsack
```

public class **MultidimensionalKnapsack**
extends Object

Constructor Index

- **MultidimensionalKnapsack**(double[], double[], double[][])
- **MultidimensionalKnapsack**(double[], double[], double[][], int, int, int)
- **MultidimensionalKnapsack**(ItemOrderedSet, double[])

Method Index

- **getBestSet**()
- **getBestTime**()
- **getBestValue**()
- **getCpuTime**()
- **getIterations**()
- **getUnPackedSet**()
- **main**(String[])
- **setMaxOuterSpan**(int)
- **solve**()

Constructors

MultidimensionalKnapsack

```
public MultidimensionalKnapsack(ItemOrderedSet itemSet,
                                double rhs[])
```

MultidimensionalKnapsack

```
public MultidimensionalKnapsack(double p[],
                                double rhs[],
                                double c[][])
```

MultidimensionalKnapsack

```
public MultidimensionalKnapsack(double p[],
                                double rhs[],
                                double c[][],
                                int p1,
                                int p2,
                                int t)
```

Methods

getBestValue

```
public final double getBestValue()
```

getIterations

```
public final int getIterations()
```

setMaxOuterSpan

```
public final void setMaxOuterSpan(int mO)
```

getBestSet

```
public ItemOrderedSet getBestSet()
```

getUnPackedSet

```
public ItemOrderedSet getUnPackedSet()
```

solve

```
public void solve()
```

getBestTime

```
public double getBestTime()
```

getCpuTime

```
public double getCpuTime()
```

main

```
public static void main(String args[])
```

Class AFIT.Alm.Knapsack.Pointer

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.Pointer
```

public class **Pointer**
extends Object
implements Comparable

Constructor Index

- **Pointer**(int, double)

Method Index

- **compareTo**(Comparable)

Constructors

Pointer

```
public Pointer(int pointer,
               double objFunction)
```

Methods

compareTo

```
public int compareTo(Comparable b)
```

Class AFIT.Alm.Knapsack.QuantityPredicate

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.QuantityPredicate
```

public class **QuantityPredicate**
extends Object
implements Serializable

Constructor Index

- QuantityPredicate()

Method Index

- execute(Object)

Constructors

QuantityPredicate

```
public QuantityPredicate()
```

Methods

execute

```
public final boolean execute(Object object)
```

Class AFIT.Alm.Knapsack.ReactiveKnapsack

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.MultidimensionalKnapsack
           |
           +----AFIT.Alm.Knapsack.ReactiveKnapsack
```

---

public class **ReactiveKnapsack**
extends MultidimensionalKnapsack

Constructor Index

- **ReactiveKnapsack**(RTSParameters, double[], double[], double[][])
- **ReactiveKnapsack**(RTSParameters, ItemOrderedSet, double[])

Method Index

- **main**(String[])

Constructors

ReactiveKnapsack

```
public ReactiveKnapsack(RTSParameters param,
                        double p[],
                        double rhs[],
                        double c[][])
```

ReactiveKnapsack

```
public ReactiveKnapsack(RTSParameters param,
```

```
ItemOrderedSet itemSet,
double rhs[])
```

Methods

main

```
public static void main(String args[])
```

Class AFIT.Alm.Knapsack.RTSParameters

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.RTSParameters
```

public class **RTSParameters**
extends Object
implements Serializable

Constructor Index

- **RTSParameters**()
- **RTSParameters**(int, int, int, int)

Method Index

- **output**(PrintWriter, RTSParameters)

Constructors

RTSParameters

```
public RTSParameters()
```

RTSParameters

```
public RTSParameters(int c,
                     int mC,
                     int r,
                     int mO)
```

Methods

output

```
public static final void output(PrintWriter out,
                                RTSParameters p)
```

Class AFIT.Alm.Knapsack.Slave

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.Slave
```

public abstract class **Slave**
extends Object
implements Serializable

Constructor Index

- **Slave**()

Method Index

- **solve**(Aircraft, ItemOrderedSet, int, int, int)

Constructors

Slave

```
public Slave()
```

Methods

solve

```
public static final boolean solve(Aircraft a,
                                  ItemOrderedSet set,
                                  int min,
                                  int max,
                                  int loopCount)
```

Class AFIT.Alm.Knapsack.UnitAlm

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.UnitAlm
```

---

public class **UnitAlm**
extends Object
implements Serializable

Constructor Index

- **UnitAlm**()

Method Index

- **getEquipment**()
- **getID**()
- **getName**()
- **getNumberOfEquipmentTypes**()
- **setEquipment**(ID[])
- **setId**(int)
- **setName**(String)
- **setNomenclature**(String)
- **setNumberOfEquipmentTypes**(int)
- **setNumberOfPassengers**(int)
- **setWeightAccompanySupplies**(double)
- **setWeightAmmo**(double)
- **setWeightNonMobilEquipment**(double)
- **setWeightNonMobilPallets**(double)

Constructors

UnitAlm

```
public UnitAlm()
```

Methods

getNumberOfEquipmentTypes

```
public final int getNumberOfEquipmentTypes()
```

getID

```
public final Integer getID()
```

getName

```
public final String getName()
```

getEquipment

```
public final ID[] getEquipment()
```

setName

```
public final void setName(String s)
```

setId

```
public final void setId(int i)
```

setNomenclature

```
public final void setNomenclature(String s)
```

setWeightAccompanySupplies

```
public final void setWeightAccompanySupplies(double w)
```

setWeightAmmo

```
public final void setWeightAmmo(double w)
```

setWeightNonMobilPallets

```
public final void setWeightNonMobilPallets(double w)
```

setWeightNonMobilEquipment

```
public final void setWeightNonMobilEquipment(double w)
```

setNumberOfPassengers

```
public final void setNumberOfPassengers(int p)
```

setNumberOfEquipmentTypes

```
public final void setNumberOfEquipmentTypes(int e)
```

setEquipment

```
public final void setEquipment(ID e[])
```

---

package AFIT.Alm.Knapsack.Reader

Class Index

- EquipmentReader
- KnapSolve
- KnapsackReader

Class AFIT.Alm.Knapsack.Reader.EquipmentR

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.Reader.EquipmentReader
```

---

public abstract class **EquipmentReader**
extends Object

Constructor Index

- **EquipmentReader**()

Method Index

- **doubleValue**(String)
- **intValue**(String)
- **main**(String[])
- **readEquipmentData**()
- **readGroupData**()
- **readUnitData**()

Constructors

EquipmentReader

```
public EquipmentReader()
```

Methods

main

```
public static void main(String args[])
```

readGroupData

```
public static final Hashtable readGroupData()
```

readUnitData

```
public static final Hashtable readUnitData()
```

readEquipmentData

```
public static final Hashtable readEquipmentData()
```

doubleValue

```
public static final double doubleValue(String s)
```

intValue

```
public static final int intValue(String s)
```

Class AFIT.Alm.Knapsack.Reader.KnapsackRe

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.Reader.KnapsackReader
```

public class **KnapsackReader**
extends Object

Constructor Index

- **KnapsackReader()**

Method Index

- **main(String[])**
- **solveKnapsack(File, File, RTSParameters)**

Constructors

KnapsackReader

```
public KnapsackReader()
```

Methods

main

```
public static void main(String args[])
```

solveKnapsack

```
public static void solveKnapsack(File kFile,
                                 File oFile,
                                 RTSParameters param)
```

Class AFIT.Alm.Knapsack.Reader.KnapSolve

```
java.lang.Object
   |
   +----AFIT.Alm.Knapsack.Reader.KnapSolve
```

public abstract class **KnapSolve**
extends Object

Constructor Index

- KnapSolve()

Method Index

- solve(int, PrintWriter, int, double[], double[], double[][])
- solveRTS(int, PrintWriter, RTSParameters, double[], double[], double[][])

Constructors

KnapSolve

```
public KnapSolve()
```

Methods

solveRTS

```
public static void solveRTS(int prob,
                            PrintWriter out,
```

```
                                   RTSParameters param,
                                   double p[],
                                   double b[],
                                   double c[][])
```

solve

```
 public static void solve(int prob,
                          PrintWriter out,
                          int maxSpan,
                          double p[],
                          double b[],
                          double c[][])
```

---

package AFIT.Alm.Packing

Interface Index

- Cargo

Class Index

- Aircraft
- BalancedContainer
- C17
- CandidateListStrategy
- Cargo2d
- Container
- Helicopter
- Move
- MoveSet
- ObjectiveFunction
- PackCanvas
- PackingCanvas
- Params
- RotateMove
- SearchThread
- SearchViewer
- Section
- SectionedAircraft
- SwapMove
- Tabu
- TranslateMove
- Vehicle
- bestMove

Class AFIT.Alm.Packing.BalancedContainer

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.Container
           |
           +----AFIT.Alm.Packing.BalancedContainer
```

public class **BalancedContainer**
extends Container

The class defines a *Container* in x y cordinate space that has methods that calculate the Center of Gravity location along the x axis

Version:
  1.1 15 FEB 1998

Author:
  Christopher A. Chocolaad Air Force Institute of Technology

Constructor Index

- **BalancedContainer**(double[], double[], int, double, double)
  Instantiates a new Balanced Container

Method Index

- **cargoCGLocationX**(Cargo[], double)
  Determines the center of gravity location on the x axis of this container with array of *Cargo c* in the current packing pattern

- **cgLocationX**(Cargo[])
  Determines the center of gravity location on the x axis of this container with array of

Cargo c in the current packing pattern

Constructors

BalancedContainer

```
public BalancedContainer(double xPoints[],
                         double yPoints[],
                         int npoints,
                         double cg,
                         double emptyWeight)
```

Instantiates a new Balanced Container

Parameters:

xPoints - Array of x cordinates that define the convex polygon representation of this container

yPoints - Array of y cordinates that define the convex polygon representation of this container

npoints - The number of vertices in the convex polygon

cg - Location of the center of gravity on the x axis when the container is empty

emptyWeight - The weight of the container when it is empty

See Also:

Container

Methods

cgLocationX

```
public double cgLocationX(Cargo c[])
```

Determines the center of gravity location on the x axis of this container with array of *Cargo c* in the current packing pattern

Parameters:

c - The cargo array to base the center of gravity location on

Returns:

The floating point location of the center of gravity on the x axis

cargoCGLocationX

```
public static double cargoCGLocationX(Cargo c[],
                                      double TotalWeight)
```

Determines the center of gravity location on the x axis of this container with array of *Cargo c* in the current packing pattern

Parameters:

c - The cargo array to base the center of gravity location on

Returns:

The floating point location of the center of gravity on the x axis

---

Class AFIT.Alm.Packing.bestMove

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.bestMove
```

public class **bestMove**
extends Object
implements Serializable

Constructor Index

- **bestMove**()

Constructors

bestMove

```
public bestMove()
```

Class AFIT.Alm.Packing.C17

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.Container
           |
           +----AFIT.Alm.Packing.BalancedContainer
                   |
                   +----AFIT.Alm.Packing.Aircraft
                           |
                           +----AFIT.Alm.Packing.SectionedAircraft
                                   |
                                   +----AFIT.Alm.Packing.C17
```

---

public class **C17**
extends SectionedAircraft

Defines a C-17 `SectionedAircraft`

Version:
1.1 15 FEB, 1997

Author:
Christopher A.Chocolaad Air Force Institute of Technology

See Also:
SectionedAircraft

Constructor Index

- C17()
  Instantiates a C17 aircraft

Constructors

C17

```
public C17()
```

Instantiates a C17 aircraft

Class AFIT.Alm.Packing.CandidateListStrategy

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.CandidateListStrategy
```

public class **CandidateListStrategy**
extends Object

Implements a Block-Random Order Scan and a Full-Random Order Scan as described in Glovers 1995 article Tabu Thresholding: Improved Seaerch by Nonmontonic Trajectories. Only the number of move sets is required, the block size can either be given or one will be selected based on move set size.

Version:
  1.0 October 31, 1997
Author:
  Christopher A.Chocolaad

Constructor Index

- **CandidateListStrategy**(int)
  Constructs the class that encapsulates the candidate list strategy For move sets less than 100, the minimum of (5,move set size) is used for the block size.
- **CandidateListStrategy**(int, int)
  Constructs the class that encapsulates the candidate list strategy

Method Index

- **getNextBROS**()
  Returns the next move to make during an Improving phase based on a Block Random Order Scan

- getNextFROS()
  Returns the next move to make during a Mixed phase Based on a Full Random Order Scan, until the move set is exhausted, then reverts to the Block Random Order Scan
- main(String[])
  test stub for the class

Constructors

CandidateListStrategy

```
public CandidateListStrategy(int m)
```

Constructs the class that encapsulates the candidate list strategy For move sets less than 100, the minimum of (5,move set size) is used for the block size. For move sets > 100, the block size equals (move set size/100)

Parameters:
m - move set size

CandidateListStrategy

```
public CandidateListStrategy(int m,
                             int bs)
```

Constructs the class that encapsulates the candidate list strategy

Parameters:
m - move set size
bs - block size

Methods

getNextBROS

```
public final int getNextBROS()
```

Returns the next move to make during an Improving phase based on a Block Random Order Scan

getNextFROS

```
public final int getNextFROS()
```

Returns the next move to make during a Mixed phase Based on a Full Random Order Scan, until the move set is exhausted, then reverts to the Block Random Order Scan

main

```
public static void main(String args[])
```

test stub for the class

All Packages Class Hierarchy This Package Previous Next Index

---

Interface AFIT.Alm.Packing.Cargo

public interface **Cargo**

Method Index

- **calculateBounds**()
- **getArea**()
- **getCentroidX**()
- **getCentroidY**()
- **getIntX**()
- **getIntY**()
- **getMaxX**()
- **getMaxY**()
- **getMinX**()
- **getMinY**()
- **getNPoints**()
- **getWeight**()
- **getXLocal**()
- **getXpoint**(int)
- **getXpoints**()
- **getYaw**()
- **getYLocal**()
- **getYpoint**(int)
- **getYpoints**()
- **height**()
- **intersectArea**(Cargo[])
- **intersectAreaAll**(Cargo[])
- **MatrixTransform**()
- **rotate**(double)
- **setCentroid**(double, double)
- **setMatrixRotate**(double)
- **setMatrixTranslate**(double, double)
- **setMatrixUnit**()
- **setYaw**(double)
- **swap**(Cargo)

- translate(double, double)
- width()

Methods

translate

```
public abstract void translate(double x,
                               double y)
```

rotate

```
public abstract void rotate(double theta)
```

swap

```
public abstract void swap(Cargo c)
```

setYaw

```
public abstract void setYaw(double y)
```

getYaw

```
public abstract double getYaw()
```

setCentroid

```
public abstract void setCentroid(double x,
                                 double y)
```

getCentroidX

```
public abstract double getCentroidX()
```

getCentroidY

```
public abstract double getCentroidY()
```

getXpoints

```
public abstract double[] getXpoints()
```

getYpoints

```
public abstract double[] getYpoints()
```

getXpoint

```
public abstract double getXpoint(int index)
```

getYpoint

```
public abstract double getYpoint(int index)
```

getXLocal

```
public abstract double[] getXLocal()
```

getYLocal

```
public abstract double[] getYLocal()
```

getNPoints

```
public abstract int getNPoints()
```

setMatrixUnit

```
public abstract void setMatrixUnit()
```

setMatrixRotate

```
public abstract void setMatrixRotate(double theta)
```

setMatrixTranslate

```
public abstract void setMatrixTranslate(double dx,
                                        double dy)
```

MatrixTransform

```
public abstract void MatrixTransform()
```

calculateBounds

```
public abstract void calculateBounds()
```

intersectArea

```
public abstract double intersectArea(Cargo c[])
```

intersectAreaAll

```
public abstract double intersectAreaAll(Cargo c[])
```

getIntX

```
public abstract int[] getIntX()
```

getIntY

```
public abstract int[] getIntY()
```

getArea

```
public abstract double getArea()
```

getWeight

```
public abstract double getWeight()
```

getMinX

```
public abstract double getMinX()
```

getMinY

```
public abstract double getMinY()
```

getMaxX

```
public abstract double getMaxX()
```

getMaxY

```
public abstract double getMaxY()
```

width

```
public abstract double width()
```

height

```
public abstract double height()
```

Class AFIT.Alm.Packing.Cargo2d

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.Cargo2d
```

public class **Cargo2d**
extends Object
implements Cargo

Variable Index

- **moveSet**
  The *MoveSet* for this cargo item
- **numTri**

Constructor Index

- **Cargo2d**(Cargo2d)
  Instantiates a new *Cargo2d* object with the same parameters as *c*

- **Cargo2d**(double[], double[], int)
  Instantitiates an new *Cargo2d* item.

Method Index

- **calculateBounds**()
  This method calculates the two dimensional bounding box of the cargo Item.
- **extentsOverlap**(Cargo)
  Return true if the bounding box overlaps *Cargo* item *c*.
- **getArea**()
  Get the area of the polygon

- **getCentroidX()**
  Get the x cordinate location of the centroid
- **getCentroidY()**
  Get the x cordinate location of the centroid
- **getIntX()**
  Returns an *int* array of the x cordinates of the vertices
- **getIntY()**
  Returns an *int* array of the y cordinates of the vertices
- **getMaxX()**
- **getMaxY()**
- **getMinX()**
- **getMinY()**
- **getNPoints()**
  Get the number of vertices in the polygon that represents the *Cargo* item
- **getNumTri()**
  Get the number of triangles.
- **getTriangles()**
  Get the *Triangle* array of this *Cargo2d*
- **getWeight()**
  Gets the weight of this cargo item
- **getXLocal()**
  Get the x cordinates of the local space
- **getXpoint**(int)
  Get the x cordinate of the vertice *index*
- **getXpoints()**
  Get the x cordinates of the vertices
- **getYaw()**
  Returns the yaw in degrees
- **getYLocal()**
  Get the y cordinates of the local space
- **getYpoint**(int)
  Get the y cordinate of the vertice *index*
- **getYpoints()**
  Get the y cordinates of the vertices
- **height()**
  The height of the bounding box of the cargo item
- **intersectArea**(Cargo2d)
  The intersection area of *this* cargo item with another cargo item *c*
- **intersectArea**(Cargo2d[])
  The intersection of this *Cargo* item with array of *Cargo* items *c*
- **intersectArea**(Cargo[])
  The intersection area of *this* cargo item with an array of*Cargo* items *c*
- **intersectAreaAll**(Cargo2d[])
  The intersection area of all cargo items in array *c*

- **intersectAreaAll**(Cargo[])
  The intersection of this *Cargo* item with array of *Cargo* items *c*
- **MatrixTransform**()
  Moves *Cargo2d* by its matrix.
- **rotate**(double)
  This method rotates the Cargo item around centroidX and centroidY by theta degrees and then updates the bounding box.
- **setCentroid**(double, double)
  Set the centroid to the location x, and location y
- **setCentroid**(Vert2d)
  Set the centroid of the cargo item to the vertice *c*
- **setMatrixRotate**(double)
  Rotate this *Cargo2d Matrix* by *theta*
- **setMatrixTranslate**(double, double)
  Translate this *Cargo2d Matrix* by *x* in the x direction and by *y* in the y direction
- **setMatrixUnit**()
  Set the transform matrix of this cargo item to the identity matrix
- **setMoveSet**(Cargo[], ObjectiveFunction, double, double)
- **setWeight**(double)
  Sets the weight of this cargo item
- **setYaw**(double)
  Set the yaw in degrees
- **swap**(Cargo)
  This method swaps the location of this Cargo item to the location of Cargo item c based on centroid position.
- **translate**(double, double)
  This method moves the Cargo item by deltaX in the x direction and deltaY in the y direction
- **updateExtents**()
  Update the cordinates of the *Traingle* array to the current location
- **width**()
  The width of the bounding box of the cargo item

Variables

- **moveSet**

public MoveSet moveSet

The *MoveSet* for this cargo item

- **numTri**

public int numTri

Constructors

Cargo2d

```
public Cargo2d(double xPoints[],
               double yPoints[],
               int npoints)
```

Instantitiates an new *Cargo2d* item.

Parameters:
xPoints - The x cordinates of the vertices
yPoints - The y cordinates of the vertices
npoints - The number of vertices

See Also:
Cargo, Tabu

Cargo2d

```
public Cargo2d(Cargo2d c)
```

Instantiates a new *Cargo2d* object with the same parameters as *c*

Parameters:
c - Cargo2d object to clone parameters from

See Also:
Cargo, Tabu

Methods

setMoveSet

```
public void setMoveSet(Cargo c[],
                       ObjectiveFunction f,
                       double minDis,
                       double maxDis)
```

width

```
public double width()
```

The width of the bounding box of the cargo item

Returns:
The width of the bounding box of this cargo item

height

```
public double height()
```

The height of the bounding box of the cargo item

Returns:
The height of the bounding box of this cargo item

setWeight

```
public void setWeight(double w)
```

Sets the weight of this cargo item

getWeight

```
public double getWeight()
```

Gets the weight of this cargo item

Returns:
Cargo Item *weight*

setCentroid

```
public void setCentroid(Vert2d c)
```

Set the centroid of the cargo item to the vertice *c*

setCentroid

```
public void setCentroid(double x,
                        double y)
```

Set the centroid to the location x, and location y

Parameters:
x - Cordinate of the centroid
y - Cordinate of the centroid

getIntX

```
public int[] getIntX()
```

Returns an *int* array of the x cordinates of the vertices

getIntY

```
public int[] getIntY()
```

Returns an *int* array of the y cordinates of the vertices

getYaw

```
public double getYaw()
```

Returns the yaw in degrees

Returns:
yaw in degrees

setYaw

```
public void setYaw(double y)
```

Set the yaw in degrees

Parameters:
y - yaw in degrees

getCentroidX

`public double getCentroidX()`

Get the x cordinate location of the centroid

Returns:
x cordinate of the centroid

getCentroidY

`public double getCentroidY()`

Get the x cordinate location of the centroid

Returns:
x cordinate of the centroid

getXpoint

`public double getXpoint(int index)`

Get the x cordinate of the vertice *index*

Parameters:
index - The index of the x cordinate
Returns:
x cordinate of vertice *index*

getYpoint

`public double getYpoint(int index)`

Get the y cordinate of the vertice *index*

Parameters:
index - The index of the y cordinate
Returns:
y cordinate of vertice *index*

getXpoints

`public double[] getXpoints()`

Get the x cordinates of the vertices

Returns:
x cordinates of the vertices

getYpoints

```
public double[] getYpoints()
```

Get the y cordinates of the vertices

Returns:
y cordinates of the vertices

getXLocal

```
public double[] getXLocal()
```

Get the x cordinates of the local space

Returns:
x cordinates of local space of the vertices

getYLocal

```
public double[] getYLocal()
```

Get the y cordinates of the local space

Returns:
y cordinates of local space of the vertices

getNPoints

```
public int getNPoints()
```

Get the number of vertices in the polygon that represents the *Cargo* item

Returns:
Number of vertices

getArea

```
public double getArea()
```

Get the area of the polygon

Returns:
The area of the polygon

getNumTri

```
public int getNumTri()
```

Get the number of triangles. This should alwas be the number of vertices minus 2

getTriangles

```
public Triangle[] getTriangles()
```

Get the *Triangle* array of this *Cargo2d*

Returns:
The triangles of the Cargo2d

setMatrixUnit

```
public void setMatrixUnit()
```

Set the transform matrix of this cargo item to the identity matrix

setMatrixRotate

```
public void setMatrixRotate(double theta)
```

Rotate this *Cargo2d Matrix* by *theta*

setMatrixTranslate

```
public void setMatrixTranslate(double dx,
                               double dy)
```

Translate this *Cargo2d Matrix* by *x* in the x direction and by *y* in the y direction

MatrixTransform

```
public final void MatrixTransform()
```

Moves *Cargo2d* by its matrix.

translate

```
public final void translate(double deltaX,
                            double deltaY)
```

This method moves the Cargo item by deltaX in the x direction and deltaY in the y direction

Parameters:
deltaX - deltaX is the distance to move the item in the x direction
deltaY - deltaY is the distance to move the item in the y direction

rotate

```
public final void rotate(double theta)
```

This method rotates the Cargo item around centroidX and centroidY by theta degrees and then updates the bounding box.

Parameters:
theta - theta is the angle in degrees to rotate the object

swap

public final void swap(Cargo c)

This method swaps the location of this Cargo item to the location of Cargo item c based on centroid position.

Parameters:
c - c is the cargo item to swap locations with

calculateBounds

public final void calculateBounds()

This method calculates the two dimensional bounding box of the cargo Item. A retangle is greated with appropriate dimensions the can be accessed by calling getBounds.

See Also:
getBounds

intersectArea

public double intersectArea(Cargo c[])

The intersection area of *this* cargo item with an array of*Cargo* items *c*

Parameters:
c - Array of cargo items to check itersection with *this* cargo item

intersectArea

public final double intersectArea(Cargo2d c)

The intersection area of *this* cargo item with another cargo item *c*

Parameters:
c - Cargo item to check for intersection

intersectArea

public double intersectArea(Cargo2d c[])

The intersection of this *Cargo* item with array of *Cargo* items *c*

Parameters:
c - The array to check for intersection

Returns:
Intersection area

intersectAreaAll

public double intersectAreaAll(Cargo c[])

The intersection of this *Cargo* item with array of *Cargo* items *c*

Parameters:
c - The array to check for intersection
Returns:
intersection area

intersectAreaAll

`public static final double intersectAreaAll(Cargo2d c[])`

The intersection area of all cargo items in array *c*

updateExtents

`public final void updateExtents()`

Update the cordinates of the *Traingle* array to the current location

See Also:
Triangle

extentsOverlap

`public final boolean extentsOverlap(Cargo c)`

Return true if the bounding box overlaps *Cargo* item *c*.

Returns:
True if the bounding box overlaps

getMinX

`public double getMinX()`

Returns:
The lowest x cordinate

getMinY

`public double getMinY()`

Returns:
The lowest y cordinate

getMaxX

`public double getMaxX()`

Returns:
The maximum x cordinate

getMaxY

`public double getMaxY()`

Returns:
The maximum y cordinate

Class AFIT.Alm.Packing.Container

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.Container
```

public class **Container**
extends Object

A convex shaped container. This container works with the the *Tabu* class in the *Packing* package. Has methods that detrmine if cargo are protuding from the container determine the bounding box of the container and the centroid of the container.

Version:
1.1 January 11, 1998

Author:
Christopher A. Chocolaad Air Force Institute of Technolgy

Constructor Index

- **Container**(double, double, double, double)
  Instantiates a new rectangular shaped two dimensional `Container` with the upper left hand corner at pointx,y with dimensions*width* and *height*
- **Container**(double[], double[], int)
  Constructs a new polyon shaped Container with cordinates (`xPoints`, `yPoints`) The container must be convex or the *protrusion* mehtod will not work correctly.

Method Index

- **calculateBounds**()
  Calculates the bounding box of the container and updates the *width* and *height*
- **getNpoints**()
  Returns the array of number of points or vertices that make up the container
- **getProtrusion**(Cargo)

The protrusion distance is calculated using P = Px + Py where Px is the distance in the x direction that `c` is from the centroid of the container and Py is the distance in the y direction `c` from the centroid of the container.

- **getXpoints**()
  Returns the array of xPoints that make up the container
- **getYpoints**()
  Returns the array of yPoints that make up the container
- **length**()
  Length of the bounding box

- **main**(String[])
  test stub for the class
- **width**()
  Width of the bounding box

Constructors

Container

```
public Container(double xPoints[],
                 double yPoints[],
                 int npoints)
```

Constructs a new polyon shaped Container with cordinates (`xPoints`, `yPoints`) The container must be convex or the *protrusion* mehtod will not work correctly.

Parameters:
xPoints - the *x* coordinates.
yPoints - the *y* coordinates.
npoints - the number of points in *xPoints* and *yPoints*
See Also:
Tabu

Container

```
public Container(double x,
                 double y,
                 double width,
                 double height)
```

Instantiates a new rectangular shaped two dimensional `Container` with the upper left hand corner at point*x,y* with dimensions*width* and *height*

Methods

getXpoints

```
public double[] getXpoints()
```

Returns the array of xPoints that make up the container

getYpoints

```
public double[] getYpoints()
```

Returns the array of yPoints that make up the container

getNpoints

```
public int getNpoints()
```

Returns the array of number of points or vertices that make up the container

length

```
public double length()
```

Length of the bounding box

Returns:
The length of the bounding box.

width

```
public double width()
```

Width of the bounding box

Returns:
The width of the bounding box.

getProtrusion

```
public double getProtrusion(Cargo c)
```

The protrusion distance is calculated using P = Px + Py where Px is the distance in the x direction that `c` is from the centroid of the container and Py is the distance in the y direction `c` from the centroid of the container. Then the protrusion distance is

squared. Returns zero if no protrusion.

Parameters:
c - cargo to check for protrusion

Returns:
The squared distance from the centroid to the protruding cargo

See Also:
Cargo

main

```
public static void main(String args[])
```

test stub for the class

calculateBounds

```
public void calculateBounds()
```

Calculates the bounding box of the container and updates the *width* and *height*

Class AFIT.Alm.Packing.Helicopter

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.Cargo2d
           |
           +----AFIT.Alm.Packing.Helicopter
```

public class **Helicopter**
extends Cargo2d

A Helicopter in two dimensional space, used to demonstrate non-convex packing

See Also:
Cargo2d

Constructor Index

- **Helicopter**()
  Instantiates a *Helicopter*

Constructors

Helicopter

```
public Helicopter()
```

Instantiates a *Helicopter*

Class AFIT.Alm.Packing.MoveSet

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.MoveSet
```

public class **MoveSet**
extends Object
implements Serializable

Moveset defines a set of moves to be made by *Cargo* item *item* for use with the *Tabu* packing heuristic.

Version:
1.1 15 FEB 1998

Author:
Christopher A. Chocolaad

Constructor Index

- MoveSet(Cargo, Cargo[], ObjectiveFunction, double, double)
  Constructs a new *Cargo* object.

Method Index

- **bestMove**()
  Move the item by an absolute best Move.
- **improvingMove**()
  Move the Cargo item by a probalistic best move
- **mixedMove**()
  Makes a random swap or rotate move

Constructors

MoveSet

```
public MoveSet(Cargo item,
               Cargo cargoArray[],
               ObjectiveFunction f,
               double minDis,
               double maxDis)
```

Constructs a new *Cargo* object.

Parameters:
item - The *Cargo* item that this moveSet will be atttached to
cargoArray - The cargoArray that *item*is a part of
f - Object Function used to evalute potential moves
minDis - The minimum distance allowed for a move
maxDis - The maximum distance allowed for a move

See Also:
Cargo2d, ObjectiveFunction, Tabu

Methods

improvingMove

```
public final boolean improvingMove()
```

Move the Cargo item by a probalistic best move

bestMove

```
public final boolean bestMove()
```

Move the item by an absolute best Move. The set is subsetof the improvingMove set

mixedMove

```
public void mixedMove()
```

Makes a random swap or rotate move

Class AFIT.Alm.Packing.ObjectiveFunction

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.ObjectiveFunction
```

public class **ObjectiveFunction**
extends Object
implements Serializable

Evaluates an array of Cargo Items and returns floating point number based on the defined objective function

Version:
1.1 15 FEB 1998

Author:
Christopher A. Chocolaad Air Force Institute of Technology

See Also:
Tabu, MoveSet'

Constructor Index

- **ObjectiveFunction**(Cargo[], Aircraft)
  Constructs a new *ObjectiveFunction*

Method Index

- **feasible**()
  Returns true if the current packing pattern is feasible
- **objFunction**()
  Evaluate the current Packing Pattern, this ignores the weights
- **objFunctionItem**(Cargo)

Evaluate posistion of an item based on current posistion using weights

- resetNorms()

  Set the norms back to a constant value

- setWeights(double, double, double, double)

  Set the weights for Ovelap penalty, Bounding Box Penalty,Protrision Penalty, and Centr of Gravity penalties

Constructors

ObjectiveFunction

```
public ObjectiveFunction(Cargo items[],
                         Aircraft a)
```

Constructs a new *ObjectiveFunction*

Parameters:

items - Array of *Cargo* items that will be used to evaluated

a - *Aircraft* to evaluate

Methods

setWeights

```
public void setWeights(double ol,
                       double bb,
                       double p,
                       double cg)
```

Set the weights for Ovelap penalty, Bounding Box Penalty,Protrision Penalty, and Centr of Gravity penalties

feasible

```
public final boolean feasible()
```

Returns true if the current packing pattern is feasible

Returns:

True if the current packing pattern is feasible

resetNorms

```
public void resetNorms()
```

Set the norms back to a constant value

objFunction

```
public final double objFunction()
```

Evaluate the current Packing Pattern, this ignores the weights

objFunctionItem

```
public final double objFunctionItem(Cargo item)
```

Evaluate posistion of an item based on current posistion using weights

---

Class AFIT.Alm.Packing.PackCanvas

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Canvas
                   |
                   +----AFIT.Alm.Packing.PackCanvas
```

public class **PackCanvas**
extends Canvas

Constructor Index

- **PackCanvas**(Aircraft, Cargo[])

Method Index

- **paint**(Graphics)

Constructors

PackCanvas

```
public PackCanvas(Aircraft a,
                  Cargo c[])
```

Methods

paint

```
public void paint(Graphics g)
```

Overrides:

paint in class Canvas

Class AFIT.Alm.Packing.Params

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.Params
```

public class **Params**
extends Object

Params is used to connect a *SearchViewer* with a *Tabu* search

See Also:
Tabu, SearchViewer

Constructor Index

- **Params**()

Constructors

Params

```
public Params()
```

Class AFIT.Alm.Packing.RotateMove

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.Move
           |
           +----AFIT.Alm.Packing.RotateMove
```

public class **RotateMove**
extends Move

Provides a set of methods to rotate a Cargo Item around the z axis

Version:
1.1 15 FEB 1998
Author:
Christopher A. Chocolaad Air Force Institute of Technology
See Also:
Move

Constructor Index

- **RotateMove**(Cargo, double)
  Constructs a new *Rotate* move for *Cargo item* that will rotate *theta* degrees around the z axis

Method Index

- **move**()
  Rotate a *Cargo object theta degrees around the z axis*
- ***unmove()***
  Rotate a Cargo object negative theta degrees around the z axis

Constructors

RotateMove

```
public RotateMove(Cargo item,
                  double theta)
```

Constructs a new Rotate move for Cargo item that will rotate theta degrees around the z axis

Parameters:

item - The Cargo item to rotate
theta - The degrees to rotate Cargo item

Methods

unmove

```
public void unmove()
```

Rotate a Cargo object negative theta degrees around the z axis

Overrides:

unmove in class Move

move

```
public void move()
```

Rotate a Cargo object theta degrees around the z axis

Overrides:

move in class Move

Class AFIT.Alm.Packing.SearchThread

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----AFIT.Alm.Packing.SearchThread
```

public class **SearchThread**
extends Thread

This class makes a thread to run a *Tabu* packing search

Version:
1.1 15 FEB 1998

Author:
Christopher A. Chocolaad Air Force Institute of Technology

Constructor Index

- **SearchThread**(Tabu, int, JCProgressMeter)
  Instantiates a new *SearchThread*

Method Index

- **run**()
  Executes the packing search
- **setToBestFound**()
  Sets the packing pattern to best found patter

Constructors

- **SearchThread**

```
public SearchThread(Tabu t,
                    int i,
                    JCProgressMeter j)
```

Instantiates a new *SearchThread*

Parameters:

t - The packing *Tabu* search that will be executed
j - The *JCProgressMeter* that indicates the search progress

Methods

setToBestFound

```
public void setToBestFound()
```

Sets the packing pattern to best found patter

run

```
public void run()
```

Executes the packing search

Overrides:

run in class Thread

Class AFIT.Alm.Packing.SearchViewer

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----AFIT.Alm.Packing.SearchViewer
```

public class **SearchViewer**
extends Thread

Constructor Index

- **SearchViewer**(Canvas, Aircraft, Cargo[], Params, FormattedTextField, FormattedTextField, FormattedTextField, FormattedTextField, FormattedTextField, FormattedTextField)

Method Index

- **draw**()
- **run**()

Constructors

SearchViewer

```
public SearchViewer(Canvas c,
                    Aircraft a,
                    Cargo car[],
                    Params p,
                    FormattedTextField f1,
                    FormattedTextField f2,
                    FormattedTextField f3,
                    FormattedTextField f4,
                    FormattedTextField f5,
```

```
FormattedTextField f6)
```

Methods

draw

```
public void draw()
```

run

```
public void run()
```

Overrides:

run in class Thread

Class AFIT.Alm.Packing.Section

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.Section
```

public class **Section**
extends Object

Defines a two dimensional geometric area inside an aircraft with parameters for max Weight and axil load

Variable Index

- **maxy**
- **minY**

Constructor Index

- **Section**(double, double, double)
- **Section**(double, double, double, String)

Variables

minY

```
public double minY
```

maxy

```
public double maxy
```

Constructors

Section

```
public Section(double length,
               double width,
               double mWeight)
```

Section

```
public Section(double length,
               double width,
               double mWeight,
               String name)
```

All Packages Class Hierarchy This Package Previous Next Index

---

Class AFIT.Alm.Packing.SectionedAircraft

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.Container
           |
           +----AFIT.Alm.Packing.BalancedContainer
                   |
                   +----AFIT.Alm.Packing.Aircraft
                           |
                           +----AFIT.Alm.Packing.SectionedAircraft
```

---

public class **SectionedAircraft**
extends Aircraft

A Sectioned Aricraft specifies an `Aircraft` in in a coordinate space that is defined by an array of `Sections`. The aircraft's geometry is centrered around a centerline of 150

Version:
 1.1 15 FEB, 1997
Author:
 Christopher A.Chocolaad Air Force Institute of Technology
See Also:
 Aircraft;, Section;

Constructor Index

- **SectionedAircraft**(Section[], int, double, double, double)
 Instantiates a sectioned Aircraft.

Constructors

- **SectionedAircraft**

```
public SectionedAircraft(Section sectionArray[],
                         int numSections,
                         double cg,
                         double emptyWeight,
                         double maxAcl)
```

Instantiates a sectioned Aircraft.

Parameters:

sectionArray - An array of `Section`
numSections - The number of `Sections` that make of the Aircraft
cg - The longitudinal loaction of the center of gravity
emptyWeight - The empty weight of the Aircraft
maxAcl - The maximum cabin load the aircraft can carry

See Also:

Aircraft;, Section;

Class AFIT.Alm.Packing.SwapMove

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.Move
           |
           +----AFIT.Alm.Packing.SwapMove
```

public class **SwapMove**
extends Move

Provides a set of methods to swap a Cargo Item with another item

Version:
1.1 15 FEB 1998

Author:
Christopher A. Chocolaad Air Force Institute of Technology

See Also:
Move

Constructor Index

- **SwapMove**(Cargo, Cargo[], int, int)
  Constructs a new *Swap* move for *Cargo item* that will swap *item* with another item between in the array *cargoArray* between the index of *minIndex* and *maxIndex*

Method Index

- **move**()
  Swaps this item with another Item
- **newSwapItem**()
  Generates a new Item to swap with this item
- **unmove**()
  Undoes the swaps between this item with another Item

Constructors

SwapMove

```
public SwapMove(Cargo item,
                Cargo cargoArray[],
                int minIndex,
                int maxIndex)
```

Constructs a new *Swap* move for *Cargo item* that will swap *item* with another item between in the array *cargoArray* between the index of *minIndex* and *maxIndex*

Parameters:

item - The *Cargo* item to rotate
theta - The degrees to rotate *Cargo item*

Methods

newSwapItem

```
public void newSwapItem()
```

Generates a new Item to swap with this item

unmove

```
public void unmove()
```

Undoes the swaps between this item with another Item

Overrides:

unmove in class Move

move

```
public void move()
```

Swaps this item with another Item

Overrides:

move in class Move

Class AFIT.Alm.Packing.Tabu

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.Tabu
```

public class **Tabu**
extends Object

Variable Index

- **objFunct**

Constructor Index

- **Tabu**(Aircraft, Cargo[], int, int)

Method Index

- **feasible**()
- **getbestValue**()
- **getcurrentValue**()
- **improvingPhase**()
- **mixedPhase**()
- **setToBestFound**()

Variables

objFunct

```
public ObjectiveFunction objFunct
```

Constructors

Tabu

```
public Tabu(Aircraft a,
            Cargo c[],
            int low,
            int high)
```

Methods

setToBestFound

```
public void setToBestFound()
```

getcurrentValue

```
public final double getcurrentValue()
```

getbestValue

```
public final double getbestValue()
```

feasible

```
public final boolean feasible()
```

mixedPhase

```
public void mixedPhase()
```

improvingPhase

```
public void improvingPhase()
```

Class AFIT.Alm.Packing.TranslateMove

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.Move
           |
           +----AFIT.Alm.Packing.TranslateMove
```

public class **TranslateMove**
extends Move

Provides a set of methods to translate a Cargo Item

Version:
1.1 15 FEB 1998

Author:
Christopher A. Chocolaad Air Force Institute of Technology

See Also:
Move

Constructor Index

- **TranslateMove**(Cargo, double, double)
  Constructs a new *TranslateMove* or *Cargo item* that will translate the item *xDis* in the x direction and *yDis* in the y direction

Method Index

- **move**()
  Moves the Cargo Item by *xDis,yDis*
- **unmove**()
  Moves the Cargo Item by *-xDis,-yDis*

Constructors

TranslateMove

```
public TranslateMove(Cargo item,
                     double xDis,
                     double yDis)
```

Constructs a new *TranslateMove* or *Cargo item* that will translate the item *xDis* in the x direction and *yDis* in the y direction

Parameters:

item - The *Cargo* item to translate
xDis - The distance to move the i>Cargo item in the x direction
yDis - The distance to move the i>Cargo item in the y direction

Methods

unmove

```
public void unmove()
```

Moves the Cargo Item by *-xDis,-yDis*

Overrides:

unmove in class Move

move

```
public void move()
```

Moves the Cargo Item by *xDis,yDis*

Overrides:

move in class Move

Class AFIT.Alm.Packing.Vehicle

```
java.lang.Object
   |
   +----AFIT.Alm.Packing.Cargo2d
           |
           +----AFIT.Alm.Packing.Vehicle
```

public class **Vehicle**
extends Cargo2d

Vehicle is a *Cargo* item with seperation contraints

Version:
1.1 15 FEB 1998

Author:
Christopher A. Chocolaad Air Force Institute of Technology

Constructor Index

- **Vehicle**(double, double, double, double)
  Constructs a new vehicle with the upper left hand corner at point *x,y* and with width and height of variables with the same name.
- **Vehicle**(Vehicle)
  Constructs a vehicle with the same dimensions of *v*

Method Index

- **clone**()
- **getOverlap**()
  The overlap of the Vehicle with other *Cargo* items
- **intersectArea**(Cargo2d[])
  The intersectArea of the *Cargo* array with this *Vehicle*

Constructors

Vehicle

```
public Vehicle(double x,
               double y,
               double width,
               double height)
```

Constructs a new vehicle with the upper left hand corner at point *x,y* and with width and height of variables with the same name.

Parameters:

x - The x cordinate of the upper left hand corner
y - The y cordinate of the upper left hand corner
width - The width of the vehicle
height - The height of the vehicle

Vehicle

```
public Vehicle(Vehicle v)
```

Constructs a vehicle with the same dimensions of *v*

Parameters:

v - Vehicles to clone

Methods

clone

```
public final Object clone()
```

Overrides:

clone in class Object

getOverlap

```
public final double getOverlap()
```

The overlap of the Vehicle with other *Cargo* items

intersectArea

```
public double intersectArea(Cargo2d c[])
```

The intersectArea of the *Cargo* array with this *Vehicle*

Overrides:
intersectArea in class Cargo2d

All Packages Class Hierarchy This Package Previous Next Index

package AFIT.Alm.triangulate

Class Index

- PointT
- Triangle
- TriangulatePolygon

Class AFIT.Alm.triangulate.PointT

```
java.lang.Object
   |
   +----AFIT.Alm.triangulate.PointT
```

public class **PointT**
extends Object

Variable Index

- **x**
  x cordinate
- **y**
  y cordinate

Variables

x

```
public double x
```

x cordinate

y

```
public double y
```

y cordinate

Class AFIT.Alm.triangulate.Triangle

```
java.lang.Object
   |
   +----AFIT.Alm.triangulate.Triangle
```

public class **Triangle**
extends Object

Triangle defines a region in cordinate space based on the vertexes of a polygon. It is to be used with the `TriangulatPolygon class`. The constructor is an `int` array the must contain the numbers of the vertex's.

See Also:
TriangulatePolygon, getTriangles

Constructor Index

- **Triangle**(int[])
  Instantiates a Triangle

Method Index

- **getVertex0**()
- **getVertex1**()
- **getVertex2**()
- **triangleIntersect**(Triangle, double[], double[], double[], double[])
  Determines if this triangle intersects another triangle using the methods described in Theodoractos and Grimsley's article *The optimal packing of arbitrarily-shaped polygons using simulated annealing and polynomial-time cooling schedules* in Computer Methods in applied mechanics and engineering
- **updateExtents**(double[], double[])
  Updates the actual position of the bounding box of the Triangle.

Constructors

Triangle

```
public Triangle(int vertexNumbers[])
```

Instantiates a Triangle

Parameters:
vertexNumbers - An int array that should contain the vertex numbers of the polygon this triangle is a part of

See Also:
getTriangles

Methods

getVertex0

```
public int getVertex0()
```

Returns:
The first vertex.

getVertex1

```
public int getVertex1()
```

Returns:
The second vertex.

getVertex2

```
public int getVertex2()
```

Returns:
The third vertex.

updateExtents

```
public final void updateExtents(double x[],
                                double y[])
```

Updates the actual position of the bounding box of the Triangle.

Parameters:
: x - is the array of x cordinates for the vertices of the polygon
: y - is the array of y cordinates for the vertices of the polygon

● triangleIntersect

```
public final double triangleIntersect(Triangle t,
                                      double tx[],
                                      double ty[],
                                      double x[],
                                      double y[])
```

Determines if this triangle intersects another triangle using the methods described in Theodoractos and Grimsley's article *The optimal packing of arbitrarily-shaped polygons using simulated annealing and polynomial-time cooling schedules* in Computer Methods in applied mechanics and engineering

Parameters:
: t - Triangle to check intersection with
: tx - `t`'s x cordinates
: ty - `t`'s y cordinates
: x - `this` triangles x cordinates
: y - `this` triangles y cordinates

Returns:
: The square of the overlap area

Class AFIT.Alm.triangulate.TriangulatePolygon

```
java.lang.Object
   |
   +----AFIT.Alm.triangulate.TriangulatePolygon
```

public class **TriangulatePolygon**
extends Object

This class triangualtes a polygon This c code version of this code came from Atul Narkhede and Dinesh Manocha's *Fast Polygon Triangulation based on Seidel's Algorithm* from the Department of Computer Science, UNC Chapel Hill. This code will triangulate a simple polygon and with holes. It is an incremental randomized algorithm whose expected complexity is $O(nlog^*n)$. In practice, it is almost linear time for a simple polygon having n vertices. The triangulation does not introduce any additional vertices and decomposes the polygon into n-2 triangles.

Version:
: choco1.0 December 1997

Author:
: Chris Chocolaad Air Force Institute of Technology

Constructor Index

- **TriangulatePolygon**(int, int[], double[][])
  This instatiates the Triangulate Polygon Class.

Method Index

- **getNumberOfTriangles**()
  Returns the number of triangle objects in the triangulated polygon
- **getTriangles**()

This returns an array of Triangles that contains the triangle vertice numbers.

Constructors

TriangulatePolygon

```
public TriangulatePolygon(int ncontours,
                          int cntr[],
                          double vert[][])
```

This instatiates the Triangulate Polygon Class. The polygon is triangulated at instatiation.

Parameters:

ncontours - This is the number of contours the polygon has the first contour is the boundary and the vertices descriping it must be anti-clockwise. All other contours are holes in the polygon and must be in clockwise order.

cntr - This is the number of points in the i'th contour. The first contour is `cntr[0]`.

vert - This the input array of vertices. The first vertice is `vert[0][0]` and `vert[0][1]` where `cert[0][0]` is the x cordinate and `vert[0][1] is the y cordinate.`

Methods

getTriangles

```
public final Triangle[] getTriangles()
```

This returns an array of Triangles that contains the triangle vertice numbers.

Returns:

The triangle objects containing the vertices of the triangles of the triangulated polygon.

See Also:

Triangle

getNumberOfTriangles

```
public final int getNumberOfTriangles()
```

Returns the number of triangle objects in the triangulated polygon

Returns:

The number of triangles

Bibliography

[1] Abdou, G. and J. Arghavani. "Interative ILP Procedured for Stacking Optimization for the 3D Palletization Problem," *International Journal of Production Research, 35*(5):1287–1304 (1997).

[2] Al-Mahmeed, Ahmed S. "Tabu Search, Combination and Integration." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 320–330, Kluwer Academic Publishers, 1996.

[3] Amiouny, Samir V., et al. "Balanced Loading," *Operations Research, 40*(2):238–245 (March 1992).

[4] Andonov, Rumen, et al. "Dynamic Programming Parallel Implementations for the Knapsack Problem," (1993). Submitted for Review to the Journal of Parallel and Distributed Computers.

[5] Arkin, Esther M., et al. "Geometric Knapsack Problems," *Algorithmica, 10*:399–427 (1993).

[6] Babayav, Djangir A., et al. "A New Knapsack Solution Approach by Integer Equivalent Aggregation and Consistency Determination," *INFORMS Journal on Computing, 9*(1):43–50 (1997).

[7] Battiti, Roberto. "Reative Search: Toward Self-Tuning Heuristics." *Modern Heuristic Search Methods* edited by V. J. Rayward-Smith, et al., 61–83, John Wiley and Sons Ltd, 1996.

[8] Battiti, Roberto and Giampietro Tecchiolli. "The Reactive Tabu Search," *ORSA Journal on Computing, 6*(2):126–140 (oct 1992).

[9] Battiti, Roberto and Giampietro Tecchiolli. "Local Search with Memory: Benchmarking RTS," *Operations Research Spectrum* (1995).

[10] Battiti, Roberto, et al. "Vector Quantization with the Reative Tabu Search." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 330–342, Kluwer Academic Publishers, 1996.

[11] Beasley, J., "OR-Library." internet, 1997. available by anonymous ftp to msemga.ms.is.ac.uk.

[12] Blazewicz, J., et al. "Using a Tabu Search Approach for Solving the Two-Dimensional Irregular Cutting Problem," *Annals of Operations Research, 41*:313–325 (1993).

[13] Blazewicz, J. and R. Walkowiak. "A Local Search Approach for Two-Dimesional Irregular Cutting," *OR Spektrum, 17*:93–98 (1995).

[14] Bohli, Paul D, et al. *Airlift Loading Model.* GRC International INC, Washington, DC, June 1996.

[15] Brosh, Israel. "Optimal Cargo Allocation on Board a Plane: A Sequential Linear Programming Approach," *European Journal of Operational Research, 8*:40–46 (1981).

[16] Brown, A. R. *Optimum Packing and Depletion.* Jeffreys and Hill Limited, 1971.

[17] Buss, Arnold H. and Kirk A. Stork. "Discrete Event Simulation on the World Wide Web Using JAVA." Working Paper introduces SimKit a small set of Java classes for creating discrete event simulation models.

[18] Cagan, Jonathan. "Shape Annealing Solution to the Constrained Geometric Knapsack Problem," *Computer Aided Design, 26*(10):763–770 (October 1994).

[19] Castelino, Diane and Nelson Stephens. "Tabu Thresholding for the Frequency Assignment Problem." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 278–297, Kluwer Academic Publishers, 1996.

[20] Chu, P.C. and J.E. Beasley. "A Genetic Algorithm for the Multiconstraint Knapsack Problem." the Mangement School, Imperial College, Working paper http://mscmga.ms.ic.ac.uk/pchu/pchu.html, January 1997.

[21] Cochard, Douglas D. and Kirk A. Yost. "Improving Utilization of Air Force Cargo Aircraft," *Interfaces, 15*:53–68 (Jan 1985).

[22] Dell'Amico, Mauro. "A New Tabu Search Approach to the 0-1 Equicut Problem." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 362–377, Kluwer Academic Publishers, 1996.

[23] Devallla, Sunil. *Development of Three-Dimensional Computer - Based Heuristic for Packing.* MS thesis, Tennessee Technological University, December 1992.

[24] Dowsland, Kathryn A. "The Three Dimensional Pallet Chart: An Analysis of the Factors Affecting the Set of Feasible Layouts for a Class of Two-Dimensional Packing Problems," *Journal of the Operations Research Society, 35*:895–905 (1984).

[25] Dowsland, Kathryn A. "Some Experiments with Simulated Annealing Techniques for Packing Problems," *European Journal of Operational Research, 68*:389–399 (1993).

[26] Dowsland, Kathryn A. and William B. Dowsland. "Packing Problems," *European Journal of Operational Research, 56*:2–14 (1992).

[27] Dowsland, Kathyrn A. "Simple Tabu Thresholding and the Pallet Loading Problem." *Meta-Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 381–405, Kluwer Academic Publishers, 1996.

[28] Dumbadze, L. G. and A. T. Tizik. "Many Dimensional Knapsack Problem of Special Ladder Structure," *Journal of Computer and Systems Science Interantional, 35*(4):614–617 (1996).

[29] Dyckhoff, Harald. "A Typology of Cutting and Packing Problems," *European Journal of*

Operational Research, *44*:145–159 (1990).

[30] E. G. Coffman, Jr., et al. "Approximation Algorithms for Bin-Packing - An Updated Survey." *Algorithm Design for Computer System Design* edited by G. Ausiello, et al., 49–106, New York: Springer-Verlag, 1984.

[31] Eilon, Samuel and Nicos Christofides. "The Loading Problem," *Management Science*, *17*(5) (January 1971).

[32] Flanagan, David. *Java in a Nutshell* (Second Edition). O'Reilly and Associates, 1997.

[33] Foley, James D., et al. *Computer Graphics Principles and Practice* (Second Edition). Addison Wesley, 1990.

[34] Gehring, H., et al. "A Computer-Based Heuristic for Packing Pooled Shipment Containers," *European Journal of Operational Research*, *44*:277–288 (1990).

[35] Gerstel, Gerald. *Cargo Loading A Proposed Approach for Maximization Space Utilization of Containers Loaded with Palletized Loads*. MS thesis, Naval Postgraduate School, Monterey CA, September 1982.

[36] Gilmor, P. C. and R. E. Gomory. "The Theory and Computation of Knapsack Functions," *Operations Research*, *14*:1045–1074 (November 1966).

[37] Gilmore, P. C. and R. E. Gomory. "A Linear Programming Approach To the Cutting Stock Problem- Part II," *Operations Research*, *11*:863–888 (November 1963).

[38] Glover, Fred. "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem," *Operations Research*, (13):879–919 (November-December 1965).

[39] Glover, Fred. "Heuristics in Integer Programming Using Surrogate Constraints," *Decision Sciences*, *8*(1):156–166 (January 1977).

[40] Glover, Fred. *Tabu Search Fundamentals and Uses* (Revised and expaned Edition). University of Colorado, 1995.

[41] Glover, Fred. "Tabu Thresholding: Improving Search by Nonmonotonic Trajectories," *ORSA Journal on Computing*, 7(4):426–442 (1995).

[42] Glover, Fred and Gary A. Kochenberger. "Critical Event Tabu Search for Multidimensional Knapsack Problems." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 407–427, Kluwer Academic Publishers, 1996.

[43] Glover, Fred and Manuel Laguna. "Tabu Search." *Modern Heuristic Techniques for Combinatorial Problems* edited by Colin R. Reeves, chapter Three, 70–141, John Wiley and Sons, 1993.

[44] Glover, Fred and Manuel Laguna. *Tabu Search*. Boston: Kluwer Academic Publishers, 1997.

[45] Glover, Fred, et al. "Solving Dynamic Stochastic Control Problems in Fianance Using Tabu Search with Variable Scaling." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 429–449, Kluwer Academic Publishers, 1996.

[46] Grable, David A. "On Random Greedy Triangle Packing," *The Electronic Journal of Combinatorics*, *4* (1997).

[47] Grignon, Pierre and Georges M. Fadel. "Fuzzy Move Limit Evaluation In Structural Optimization," *American Institute of Aeronautics and Astronautics* (1994). Clemson University.

[48] Hammer, Peter L. and Jr. David J. Rader. *Efficient Methods for Solving Quadratic 0-1 Knapsack Problems.* RRR 40-94, Rutgers University, November 1994.

[49] Han, Ching Ping, et al. "A Heuristic Approach to the Three Diminsional Cargo-Loading Problem," *Interantional Journal of Production Research*, *27*(5):757–774 (1989).

[50] Hanafi, S., et al. "Comparison of Heuristics for the 0-1 Multidimensional Knapsack Problem." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 449–465, Kluwer Academic Publishers, 1996.

[51] Horstmann, Cay S. and Gary Cornell. *Core Java, One.* Sun Microsystems, 1997.

[52] Horstmann, Cay S. and Gary Cornell. *Core Java, Two.* Sun Microsystems, 1998.

[53] Huebner, Walter F. "Load Planning, Rapid Mobilization and the Computer," *Air Force Journal of Logistics*, *6*(1):22–24 (Winter 1982).

[54] III, Napoleon Bonaparte Nelson. *A Container Stuffing Algorithmfor Rectangular Solids When Voids May Be Required.* MS thesis, Naval Postgraduate School, Monterey CA, September 1979. .

[55] Ingargiola, Giorgio and James F. Korsh. "An Algorithm for the Solution of the 0-1 Loading Problems," *Operations Research*, *23*(6):1110–1119 (November 1975).

[56] Inoue, Jun-Ichi. "Statistical mechanics of the multi-constraint continous knapsack problem," *IOP*, *30*:1047–1057 (jul 1996).

[57] Ivancic, Nancy J. *An Integer Programming Based Heuristic Approach To The Three Dimensional Packing Problem.* MS thesis, Case Western Reserve University, August 1988.

[58] Khuri, Sami, et al., editors. *The Zero/One Multiple Knapsack Problem and Genetic Algorithms*, ACM Press, 1993.

[59] Khuri, Sami, et al. "The Zero/One Multiple Knapsack Problem and Genetic Algorithms." To appear in the ACM Symposiom of Applied Computation (SAC'94) proceedings, 1994.

[60] Laguna, Manuel and Fred Glover. "Bandwidth Packing: A Tabu Search Approach," *Management Science*, *39*(4):492–500 (1993).

[61] Larsen, Ole and Gert Mikkelsen. "An Interactive System for the Loading of Cargo Aircraft," *European Journal of Operational Research*, *4*:367–373 (1980).

[62] Lengauer, Thomas. *Combinatorial Algorithms for Integrated Circuit Layout*. New York: John Wiley and Sons, 1990.

[63] Li, Keqin and Kam Hoi Cheng. "Heuristic Algorithms for On-Line Packing in Three Dimensions," *Journal of Algorithms*, *13*:589–605 (1992).

[64] Linden, Peter van der. *Just Java and Beyond*. Sun Microsystems, 1998.

[65] Lokketangen, Arne and Fred Glover. "Probabilistic Move Selection in Tabu Search for Zero-One Integer Programming Problems." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 278–297, Kluwer Academic Publishers, 1996.

[66] Magent, Michael A. *Combining Neural Networks and Tabu Search In A Fast Neural Network Simulation for Combinatorial Optimization*. PhD dissertation, Lehigh, October 1996.

[67] Martello, Silvano and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. New York: John Wiley and Sons, 1990.

[68] Martin-Vega, Louis A. "Aircraft Load Planning and the Computer: Description and Review," *Computers and Industial Engineering*, *9*(4):357–369 (1985).

[69] Moser, Martin, et al. "An Algorithm for the Multidimensional Multiple-Choice Knapsack Problem," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 582–589 (1997).

[70] Narkhede, Atul and Dinesh Manocha. *Fast Polygon Triangulation Based on Seidel's Algorithm*. Implementation Report, UNC Chapel Hill, Department of Computer Science, 1994.

[71] Naughton, Patrick. *Java Handbook*. McGraw-Hill, 1996.

[72] Naval Computer and Telecommunications Station, Washington, DC. *Windows Airlift Loading Model*, April 1997.

[73] NG, Kevin Y. K. "A Multicriteria Optimization Approach To Aircraft Loading," *Operations Research*, *40*(6):1200–1205 (1992).

[74] Niar, Smail and Arnaud Freville. "A Parallel Tabu Search Algorithm for the 0-1 Multidimensional Knapsack Problem." *Proceedings: 11th International Parallel Processing Symposium April 1-5*, edited by International Parallel Processing Symposium. 512–516. Los Alamitos, California: IEEE Computer Society Press, 1997.

[75] Osman, Ibrahim H. and James P. Kelly. "Meta-Heuristics: An Overview." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 1–21, Kluwer

Academic Publishers, 1996.

[76] Preparata, Franco P. and Michael Ian Shamos. *Computational Geometry An Introduction.* New York: Springer-Verlag, 1985.

[77] RHEE, Wansoo T. and Michel Talagrand. "Multidimensional Optimal Bin Packing with Items of Random Size," *Mathematics of Operations Research, 16*(3):490–503 (August 1991).

[78] Roskam, Jan. *Airplane Flight Dynamics and Automatic Flight Controls, 1.* Route 4, Box 274. Ottaw Kansas 66067: Roskam Aviation and Engineering Corporation, 1979.

[79] Sadeh, Norman M. and Sam R. Thangiah. "Learning to Recognize (Un)Promising Simulated Annealing Runs: Efficient Search Procedures for Job Shop Scheduling and Vehicle Routing." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 278–297, Kluwer Academic Publishers, 1996.

[80] Schepers, Jorg. *An Exact Algorithm for Genral Orthogonal N-Dimensional Knapsack Problems.* Technical Report 97-258, Center for Parallel Computing, University of Cologne, 1996.

[81] Scholl, Armin, et al. "BISON: A Fast Hybrid Procedure for Exactly Solving the One-Deimensioanl Bin Packing Problem," *Computers and Operations Research, 24*(7):627–645 (1997).

[82] Sedgewick, Robert. *Algorithms in C++.* New York: Addison-Wesley Publishing Company, 1990.

[83] Sondergeld, Lutz and Stefan Vob. "A Star Shaped Diversification Approach in Tabu Search." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 278–297, Kluwer Academic Publishers, 1996.

[84] Stasko, John T. *Three Dimensional Computation Visulization.* GIT-GVU-92-20, Georgia Institute of Technology, 1992.

[85] Szykman, S. and J. Cagan. "A Simualted Annealing Approach to Three Dimensional Component Packing," *ASME Journal of Mechanical Design, 117*:308–314 (1995).

[86] Szykman, S. and J. Cagan. "Synthesis of Optimal Nonorthogonal Routes," *ASME Journal of Mechanical Design, 118*:419–424 (1996).

[87] Szykman, S. and J. Cagan. "Constrained Three Dimensional Component Layout Using Simulated Annealing," *ASME Journal of Mechanical Design, 119*:28–35 (1997).

[88] Szykman, Simon and Jonathon Cagan. "Automated Generation of Optimally Directed Three Dimensional Component Layouts." *Advances in Design Automation 1993: Proceedings of the 19th ASME Design Automation Conference.* 19–22. September 1993.

[89] Tatineni, Sayee and Georges M. Fadel. *Coupling Through Movelimits in Multi-Disciplinary*

Optimization. MS thesis, Clemson University.

[90] Theodoracatos, Vassilios E. and James L. Grimsley. "The Optimal Packing of Arbitrarily-Shaped Polygons Using Simulated Annealing and Polynomial-Time Cooling Schedules," *Computer Methods in Applied Mechanics and Engineering, 125*:53–70 (1995).

[91] Thomas, Clive, et al. "Aircraft Loading Problem." *Airline Group of the International Federation of Operation Research Societies Annual Symposium.* 123–143. AGIFORS, 1994.

[92] Toulse, Michel, et al. "Communication Issues in Designing Cooperative MultiThreaded Parallel Searches." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 503–522, Kluwer Academic Publishers, 1996.

[93] Tseng, Fan T. "A Study of Algorithms for Selecting R Best Elemnts Form an Array." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 523–535, Kluwer Academic Publishers, 1996.

[94] Udy, Jerry L., et al. "Computation of Interferences Between Three-Dimensional Objects and the Optimal Packing Problem," *Advances in Engineering Software, 10*(1):8–14 (1988).

[95] USAF Studies and Analysis Agency. *Airlift Loading Model, Analysts Manual* (Version 5.0 Edition), October 1994.

[96] Valls, Vicente. "A Modified Tabu Thresholding Approach for the Generalized Restricted Vertex Colouring Problem." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 537–553, Kluwer Academic Publishers, 1996.

[97] Vycital, Gary C. *Airlift of Army General Purpose Forces (HQ USAF SABER SIZE-Army Study).* Technical Report, HQ USAF, April 1981.

[98] Wodziak, John R. and Georges M. Fadel. "Packing and Optimizing the Center of Gravity Location Using a Genetic Algorithm." Clemson University, Working paper.

[99] Woodruff, David L. "Chunking Applied to Reactive Rabu Search." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 555–569, Kluwer Academic Publishers, 1996.

[100] Wright, Mike B. and Richard C. Marett. "A Preliminary Investigation Into the Performance of Heuristic Search Methhods Applied to Compound Combinatorial Problems." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 300–317, Kluwer Academic Publishers, 1996.

[101] Yamamoto, Akira, et al. "Asymmetric Neural Network and Its Application to Knapsack Problem," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 78*(3):300–305 (mar 1995).

[102] Yost, Kirk and Ronald W. Hare. *Airlift Estimation for MAC Cargo Aircraft.* AFLMC-LY870104 AD-B125 174, Gunter AFB, AL 36114: Air Force Logistics Management

Center, August 1988.

[103] Zachariasen, Martin and Martin Dam. "Tabu Search on the Geometric Traveling Salesman Problem." *Meta - Heuristics Theory and Applications* edited by Ibrahim H. Osman and James P. Kelly, 571–587, Kluwer Academic Publishers, 1996.

Vita

Christopher Anthony Chocolaad was born in West Palm Beach, Florida, on 21 April 1972, the son of Geraldine Gladwin Chocolaad, and Ronald Frank Chocolaad. In June 1990 he graduated from Cardinal Newman High School and entered the United States Air Force Academy. On 1 June 1994 he graduated from the United States Air Force Academy, accepting a commission to the United States Air Force and earning a Bachelor of Science in Aeronautical Engineering. While on activity duty he served one tour with the 51st Fighter Wing, Osan Air Base, Republic of Korea as Chief of the Manpower Office. He entered the Graduate School of Engineering, Air Force Institute of Technology August 15 of 1996.

Permanent Address: 1602 Pine Tree Lane #35

Dayton, Ohio 45449

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1998	Master's Thesis

4. TITLE AND SUBTITLE
Solving Geometric Knapsack Problems Using Tabu Search Heuristics

5. FUNDING NUMBERS

6. AUTHOR(S)
Christopher A. Chocolaad, Lieutenant, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
2750 P Street
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOR/ENS/98M-05

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFSAA/SAG
1570 Air Force Pentagon
Washington DC, 20330-1570

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
An instance of the geometric knapsack problem occurs in air lift loading where a set of cargo must be chosen to pack in a given fleet of aircraft. This paper demonstrates a new heuristic to solve this problem in a reasonable amount of time with a higher quality solution then previously reported in literature. We also report a new tabu search heuristic to solve geometric knapsack problems. We then employ our novel heuristics in a master-slave relationship, where the knapsack heuristic selects a set of cargo and the packing heuristic determines if that set is feasible. The search incorporates learning mechanisms that react to cycles and thus is robust over a large set of problem sizes. The new knapsack and packing heuristics compare favorably with the best reported efforts in the literature. Additionally, we show the JAVA language to be an effective language for implementing the heuristics. The search is then used in a real world problem of determining how much cargo can be packed with a given fleet of aircraft.

14. SUBJECT TERMS
Air Lift Loading Problem;Geometric Knapsack Problem;Tabu Search Heuristic;Packing Problem

15. NUMBER OF PAGES
179

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL